Universidad Politécnica de Madrid

Facultad de Informática

Proyecto Fin de Carrera

Metodologías de desarrollo ágil aplicadas al análisis de usabilidad en plataformas web

Autor: *Roberto Costumero Moreno*
Tutores: *Ricardo Imbert Paredes*
Víctor Robles Forcada

Madrid, 12 de Julio de 2013

A mis padres, que me lo han dado todo.

A Rocío, por ser simplemente tú.

You can't connect the dots looking forward, you can only connect them looking backwards.

Steve Jobs

You have to expect things of yourself before you can do them.

Michael Jordan

Agradecimientos

Desde pequeño mis padres me enseñaron que hay que dar las gracias por aquello que se tiene y estos años en la Facultad me han demostrado que la gratitud es uno de los aspectos fundamentales de la vida y que, dando las gracias, uno se siente más feliz y mejor consigo mismo.

Por esto, quiero dar las gracias a todas las personas que de alguna forma u otra han contribuido a que sea tal y como soy y que han hecho posible que este trabajo sea una realidad. Pero, particularmente, quiero dedicar este Proyecto a las personas que menciono a continuación:

A mis padres, Eusebio y Mª del Carmen, porque si no fuera por vosotros no estaría aquí, porque me lo habéis dado todo, porque os debo la vida. Por todo lo que habéis hecho por mí y lo que hacéis cada día. Gracias de todo corazón. Os quiero.

A mi abuelo Eusebio, que sigues muy presente en mi vida. *A mi abuela Seve*, porque sigo escuchando ese "corazón" que me decías cuando llegaba a casa. *A mi abuelo Federico*, por enseñarme a silbar e inculcarme la pasión por la música clásica. Descansad en paz.

A mi abuela Milagros, por ser siempre tan activa y moderna, enseñándome que la edad no cuenta y que siempre hay algo nuevo que aprender.

A mi primo hermano Fernando, que siempre has sido un hermano para mi pese a la distancia, por todos los momentos que hemos vivido desde pequeños. *A mi primo David*, porque es el futuro de la familia y un ahijado perfecto. *A mi prima Marta*, porque siempre ha sido una referencia para mí.

A mis primos y mis tíos, porque la familia siempre está para apoyarse y que a pesar de que no siempre se puede tener el mismo trato con todos, os quiero. En especial *a mi tío Fede*, por compartir sus aficiones conmigo y tratarme siempre como a un adulto más.

A mi tía Amparo, por el trato que siempre me ha brindado. *A mi tía abuela Amparo*, por su marcha andaluza y porque sabes que te quiero mucho y siempre te llevo en mi corazón.

A mi nueva familia, por haberme aceptado como a uno más desde el principio.

A Rocío, por ser simplemente tú, por todo el cariño que me demuestras y por ayudarme y apoyarme cada día en todas mis decisiones. Por ser el pilar que sustenta mi vida y por pensar en nuestro futuro juntos y querer compartir cada instante de nuestras vidas a mi lado. Te quiero y siempre te querré.

A Marta, porque me enseñaste que vivir para los demás es duro, pero muy gratificante. Por darme el mayor regalo de mi vida y pasar de ser una gran amiga a mi hermanita. Te quiero.

A toda la gente de la FI, porque al final habéis sido mi familia en Madrid. En general, quería dar las gracias a toda la gente de *ACM* y *DAFI*, por todo lo que me habéis enseñado y los buenos momentos que hemos compartido.

A Paola, que siempre me has enseñado que la perserverancia es muy importante. Por todos los buenos momentos que hemos pasado juntos y ser ese gran apoyo en los malos.

A Héctor, porque desde el principio congeniamos. Por todas las ideas locas que compartimos y que intentamos llevar a cabo, que algún día haremos realidad y, sobre todo, por aguantarme todo este año.

A Iñaki, por aguantarme estos años y encima querer compartir piso. Por todos los momentos frikis que hemos compartido y por encima de todo, nuestras discusiones que han enriquecido nuestra amistad.

A Sergio, que me has sufrido este año más que nunca. Porque tus ganas de aprender no hayan mermado y sigamos disfrutando de grandes momentos.

A Álvaro, por compartir este proyecto en el que tanto hemos "sufrido".

A Bea, por haberme demostrado que se puede confiar en poco tiempo en alguien casi sin conocerle.

A Cherni, ese gran maestro que me enseñó ACM y me ha cambiado la forma de pensar en tantas cosas.

A Vila, porque no olvidaré tantos momentos extraños y nuestras conversaciones frikis.

A Antonio, por su tesón, su buen estar y ser tan voluntarioso. Porque repitamos un viaje a CP Europe tan entretenido.

Al resto de gente de DAFI, Adri, Medra, Diana, Mon... por enseñarme ese otro mundo dentro de la Facultad.

A Pepe, porque sin ti el laboratorio no sería el mismo.

A la gente que he conocido en mis años en Madrid, en especial *a Félix*, porque intentamos conquistar el mundo antes de tiempo y casi lo conseguimos desde el Marqués.

A Elena, porque compartimos piso dos años inolvidables y siempre me has tenido en tan alta estima. Que no perdamos este contacto.

A Alex, como no podía ser de otra forma, por todas las experiencias e ideas que hemos tenido durante los cinco años que compartimos piso y todo lo que hemos sufrido juntos en la FI, que ha merecido mucho la pena.

Porque en Ávila, mi ciudad natal, siempre ha habido gente muy importante en mi vida. Gracias a todos.

A Marta Ro, porque nuestra amistad tardó en llegar pero ha sido inmejorable. Por estar ahí en los buenos y, sobre todo, en los malos momentos. Por esas copas nocturnas intentando desahogarnos y contarnos toda nuestra vida. Se te echa mucho de menos.

A Marta Ferrer, porque hemos compartido casi toda nuestra vida. Porque siempre nos hemos preocupado el uno del otro y hemos compartido tantos momentos juntos. I miss you, but I know you'll be very happy in the US. Take care.

A Patry, por ser siempre esa madraza que se preocupa por todos. Con épocas de más y épocas de menos contacto, pero siempre has estado ahí. Espero que disfrutes de tu nueva aventura en los EEUU. Cuídate mucho.

A Miguel, por todas nuestras conversaciones para arreglar la política y el mundo en general. Porque siempre me has enseñado que, a veces, ser imparcial es lo mejor.

A Don Jesús, mi profesor de la ESO, por enseñarme que ser exigente con uno mismo es el camino al éxito.

A Marisa, que en paz descanse, por todo el afecto y lo bien que me trató siempre.

A Juan Salinero, porque siempre llevo conmigo la pelota de ping-pong que me diste antes de entrar al colegio, que me recuerda lo importante que es no olvidar los buenos momentos.

Por supuesto, agradecer a toda la gente con la que he trabajado en este proyecto y en otros a lo largo de estos años de universidad, y a mis profesores, que me han enseñado tanto.

A Víctor, por confiar tanto en mí desde el principio y mostrarme el maravilloso entramado universitario.

A Ricardo, por su confianza ciega otorgándome la oportunidad de gestionar al equipo.

A Xavi, por enseñarme el mundo de la usabilidad y el diseño de las interfaces de usuario, que desde el principio me ha parecido apasionante.

A Nelson, porque no sé cómo, siempre consigue que piense más allá y que observe el resto de perspectivas del mundo.

Al resto de profesores comprometidos por hacer de ésta una universidad mejor, *Marinela, Pilar, Manuel, María, Nik, Marisa...*

A las secretarias de Decanato, porque hacen una labor increíble y casi nunca se las tiene en cuenta. *A Cristina y Ana* de Decanato, *a María*, *a Mª Jesús* y *a Mara y Coro* de los Vicedecanatos y, en especial, *a Elena*, porque siempre tenía una sonrisa para los alumnos, te deseo lo mejor y espero que puedas volver algún día.

A la Unidad de Calidad, por todo su trabajo. En especial a *Sara*, por creer en la FI para este proyecto; *a Paco*, por su implicación y su constante afán por aprender más de temas informáticos, y *a Conchi*, por todo lo que se ha pegado intentando explicarnos a fondo las necesidades técnicas del mundo del seguimiento de las titulaciones.

Por último, *a Juanlu y Trini*, de Servicios de Planificación de Sistemas de Información, por todo su esfuerzo y por enseñarnos en tan poco tiempo cómo funciona la infraestructura informática de la UPM.

GRACIAS a todos los que he nombrado y a los que no, que siempre se queda alguien en el tintero, porque si no, no sería como soy ni hubiera llegado hasta aquí.

Resumen

El desarrollo de este Proyecto Fin de Carrera se ha enmarcado dentro del diseño e implementación de una plataforma, denominada Gauss, para la gestión del seguimiento de la calidad en las titulaciones de Grado y Máster de la Universidad Politécnica de Madrid (UPM).

Esta aplicación ha sido desarrollada bajo el liderazgo del Vicerrectorado de Estructura Organizativa y Calidad (VEOC), con el apoyo del personal de la Unidad de Calidad de dicho Vicerrectorado y del Servicio de Planificación de Sistemas de Información (SPSI) del Vicerrectorado de Nuevas Tecnologías y Comunicación de la UPM, así como de los tutores del proyecto y otras personas que han formado parte del equipo de desarrollo a cargo del autor, por parte de la Facultad de Informática de la Universidad.

El Proyecto se centra en el estudio del uso de las prácticas ágiles de desarrollo para la realización de los análisis de usabilidad y la gestión del diseño e implementación de la mencionada plataforma.

Se pretende sentar las bases del uso de las prácticas ágiles dentro del proceso de diseño y estructuración de los análisis de usabilidad y de la implementación en proyectos en los que se realiza un diseño centrado en el usuario, donde la primera impresión del usuario debe ser positiva y la capacidad de retomar el uso de la plataforma desarrollada tras un largo tiempo no suponga un impedimento extra en su normal funcionamiento.

El uso de las prácticas ágiles se ha aplicado tanto al desarrollo de la plataforma como a la gestión de los hitos alcanzables para que a lo largo de las diversas iteraciones el producto obtenido fuera lo más estable posible, desarrollando y evaluando diversas partes de la plataforma.

Finalmente, se ha analizado el impacto que pueden tener este tipo de prácticas, sus puntos fuertes y débiles en este tipo de proyectos, así como posibles puntos de mejora y posibles líneas futuras de este proyecto.

Palabras clave: Prácticas de desarrollo ágil, análisis de usabilidad, diseño centrado en el usuario, prototipo, Scrum.

Índice general

Índice de figuras

Índice de tablas

Acrónimos

CLI Command Line Interface

FTP File Transfer Protocol

GUI Graphical User Interface

HCD Human Centered Design

IT Information Technologies

JIT Just-In-Time

PFC Proyecto Fin de Carrera

RUP Rational Unified Process

SPSI Servicios de Planificación de Sistemas de Información

TDD Test-Driven Development

UI User Interface

UPM Universidad Politécnica de Madrid

UX User eXperience

UXD User eXperience Design

VEOC Vicerrectorado de Estructura Organizativa y Calidad

WUI Web User Interface

XP eXtreme Programming

CAPÍTULO 1

Introducción

1.1. Introducción

En su origen, todos los proyectos necesitan de un análisis previo de los requisitos y las necesidades para afrontar, de la manera más óptima, los distintos pasos que deben realizarse a la hora de desarrollar satisfactoriamente el producto final.

En este caso, se parte de la experiencia de una plataforma previa que fue desarrollada por una empresa externa a la Universidad, que dejó de tener soporte, por lo que era necesario el desarrollo de una nueva plataforma. Esta experiencia previa ha sido útil tanto para detectar los puntos fuertes de la plataforma, como para detectar las debilidades relacionadas con su diseño que nos han permitido centrarnos mejor en realizar una plataforma útil, completa y accesible.

Debido a que la plataforma previa adolecía de notables carencias en cuanto a su usabilidad y al carácter fuertemente interactivo de la misma, se ha considerado relevante la realización de un completo análisis de usabilidad con los usuarios finales de la plataforma, que son los profesores coordinadores de las asignaturas de los distintos títulos impartidos por la UPM, así como Personal de Administración y Servicios, para realizar un enfoque más conciso de la plataforma.

El enfoque que se ha otorgado a estos análisis de usabilidad ha sido el conocido como diseño centrado en el usuario[1], mediante el cual el desarrollo de un sistema informático se realiza encadenando diversas iteraciones comenzando con prototipos en papel de baja fidelidad de la interfaz de usuario[2], pudiendo comprobar la interacción y las reacciones de los usuarios de la plataforma final para que, de esta forma, la experiencia del usuario[3] sea la mejor posible.

Dada la necesidad de evaluar los requisitos y de realizar con usuarios finales todos los pasos del proceso del diseño centrado en el usuario, además de realizar tareas de diseño e implementación de la plataforma, se valoró la necesidad de una gestión ágil y bien documentada del proyecto para que su desarrollo fuera lo más fluido posible, debido a los estrictos plazos establecidos para su finalización y puesta en producción.

[1] del inglés *Human Centered Design* - HCD
[2] del inglés *User Interface* - UI
[3] del inglés *User Experience* - UX

Es por ello que, el autor, como líder del desarrollo y de los análisis de usabilidad, ha estudiado las distintas alternativas de gestión de proyectos para valorar las más adecuadas con el objetivo de lograr los mejores resultados, sin que las actividades de gestión supusieran un obstáculo en el normal desarrollo del proyecto.

No obstante, se entiende que la gestión es una parte muy imporante dentro de los proyectos de las Tecnologías de la Información[4], ya que es una parte fundamental de comunicación y organización dentro de un equipo de trabajo, especialmente en un equipo multidisciplinar como es el caso.

Como parte de la evaluación que se hizo de los distintos sistemas de gestión de proyectos, se optó por el uso de las denominadas metodologías Ágiles, siguiendo el *Agile Manifesto* [Agi13], que han proporcionado al proyecto una mayor versatilidad y agilidad en su desarrollo, permitiendo realizar en cortas iteraciones temporales un desarrollo incremental y centrado en el usuario, para obtener resultados visibles al final de cada iteración.

Debido al ímpetu de buscar nuevos usos de diversas tecnologías y prácticas, el autor ha decidido adaptar para este proyecto una versión de las prácticas ágiles aplicándolas especialmente en el ámbito del desarrollo de los análisis de usabilidad.

A lo largo del libro se desarrollarán los distintos apartados en los que se verá el estado del arte, las distintas prácticas ágiles existentes con sus puntos fuertes y débiles, aquellas que se han usado y con qué fin se ha hecho, así como el impacto que ha tenido el uso de estas prácticas dentro del desarrollo de la plataforma.

En el **Capítulo 2** se sientan las bases de los conocimientos básicos en los que se centra este Proyecto Fin de Carrera y se muestra un amplio abanico de métodos y prácticas del desarrollo de software. En concreto, el capítulo comienza ilustrando los distintos tipos de ciclos de vida tradicionales de desarrollo de software, explicando posteriormente las novedosas prácticas ágiles y, finalmente, profundizando en los conceptos relacionados con los análisis de usabilidad.

En el **Capítulo 3** se realiza un completo análisis de la situación en la que se enmarca el Proyecto Fin de Carrera, explicando por qué se realiza el diseño de esta nueva plataforma, sus requisitos y las necesidades que cubre, así como las prácticas ágiles utilizadas durante su desarrollo, incidiendo en su adaptación al método de trabajo diseñado para la realización de este proyecto.

En el **Capítulo 4** se muestran las conclusiones obtenidas de este Proyecto Fin de Carrera y las ventajas y desventajas surgidas por el uso del método propuesto.

En el **Capítulo 5** se exponen las posibles líneas futuras del proyecto y cómo podría evolucionar.

[4]del inglés *Information Technologies* - IT

1.2. Objetivos

Los objetivos del Proyecto Fin de Carrera son los siguientes:

- Conocer los distintos tipos de metodologías de desarrollo existentes centrando el estudio en las prácticas y técnicas ágiles tales como:
  - **Scrum** - Una de las metodologías de desarrollo ágil más extendidas. No es una metodología en sí, sino más bien un conjunto de recomendaciones.
  - **Test-driven development (TDD)** - Esta práctica de desarrollo de software gestiona los proyectos de tal forma que lo primero en implementar son las pruebas del sistema para, posteriormente, generar el código que satisface dichas pruebas.
  - **Extreme programming (XP)** - Una práctica muy similar a Scrum cuya principal característica es la libertad en alguno de los cuatro factores (coste, tiempo, calidad y alcance) por parte del equipo de desarrollo. Mientras que el resto de los parámetros los suele fijar el cliente.
- Conocer el *Agile Manifesto* y comprender sus diferencias sobre metodologías de desarrollo clásicas.
- Analizar los puntos fuertes y débiles de este tipo de prácticas en el desarrollo de proyectos software.
- Vincular el uso de las prácticas seleccionadas con el desarrollo de análisis de usabilidad y el diseño centrado en el usuario de una plataforma web.
- Diferenciar los conceptos diseño centrado en el usuario, interfaz de usuario y experiencia del usuario.
- Analizar las ventajas y la eficiencia en la gestión del proyecto aportadas por este tipo de prácticas.
- Emitir conclusiones sobre el desarrollo de los prototipos y los estudios de usabilidad llevados a cabo sobre la plataforma, incidiendo en los puntos en los que las prácticas ágiles han supuesto un avance en el desarrollo del proyecto.

CAPÍTULO 2

Estado del arte

Al final de este capítulo se podrá tener una visión global de los distintos tipos de metodologías de diseño y de los ciclos de vida existentes en el desarrollo de productos software, así como de cómo se adaptan éstos en el mercado con los análisis de usabilidad realizados en el desarrollo de proyectos web en particular.

Desde principios de los años 70 cuando se definió el término de Ingeniería del Software, el desarrollo de software ha estado principalmente guiado por unos ciclos de desarrollo tradicionales donde todo proyecto tenía unas pautas concretas y unos plazos determinados establecidos y, al terminar su desarrollo, el proyecto se daba por terminado quedando en fase de mantenimiento.

A pesar de que la definición del término Ingeniería del Software no se realizó hasta los años 70, ya en la década de 1960 se realizaban desarrollos de software siguiendo unas pautas que definían una metodología funcional para grandes empresas [Alo05].

A principios del siglo XXI el mercado del software ha ido evolucionando y ha sido necesario introducir nuevos tipos de metodologías de desarrollo que se adaptaran mejor a la forma de trabajar en las empresas, así como a las exigencias de unos clientes que cada vez exigen productos con un mayor número de características en plazos más cortos de tiempo.

Por ello, surgieron las prácticas de desarrollo ágil que basan su filosofía en la realización del proyecto junto al cliente en iteraciones de desarrollo cortas (de 15 días ó 1 mes).

2.1. Ciclos de vida y prácticas tradicionales

A continuación se muestran los distintos tipos de ciclos de vida del desarrollo tradicionales que vienen utilizándose desde principios de los años 60.

2.1.1. Modelo en cascada

El modelo de desarrollo en cascada es el ciclo de vida más tradicional desde el punto de vista de la Ingeniería del Software. Como metodología tradicional enfoca el desarrollo del software de manera secuencial, de forma que el proceso evoluciona en una única dirección sin retroceder hasta terminar el proyecto, quedando éste en fase de mantenimiento y mejora.

Las fases del ciclo de vida en cascada son las siguientes:

1. **Análisis de requisitos**

   Se realiza un análisis de los requisitos del proyecto, tanto desde el punto de vista técnico como económico, centrándose especialmente en los requisitos solicitados. Esta fase también es conocida por otros autores como Ingeniería de Requisitos.

2. **Diseño**

   Durante esta fase se realizan las distintas etapas del diseño del software, realizando los diseños de alto nivel y detallado.

3. **Codificación**

   Esta fase engloba toda la tarea de implementación del sistema en un sistema informático, utilizando los lenguajes de programación y tecnologías más adecuadas, según el análisis y el diseño realizados anteriormente.

4. **Pruebas**

   Posteriormente, se realizan las pruebas del sistema, generando pruebas unitarias para cada uno de los módulos implementados, así como pruebas de integración entre los distintos módulos que componen el producto.

5. **Mantenimiento**

   Para finalizar el desarrollo del producto, éste queda en fase de mantenimiento, donde se detectan los posibles *bugs* y se proponen y realizan mejoras al sistema. La mayor parte del ciclo de vida del software se dedica a su mantenimiento.

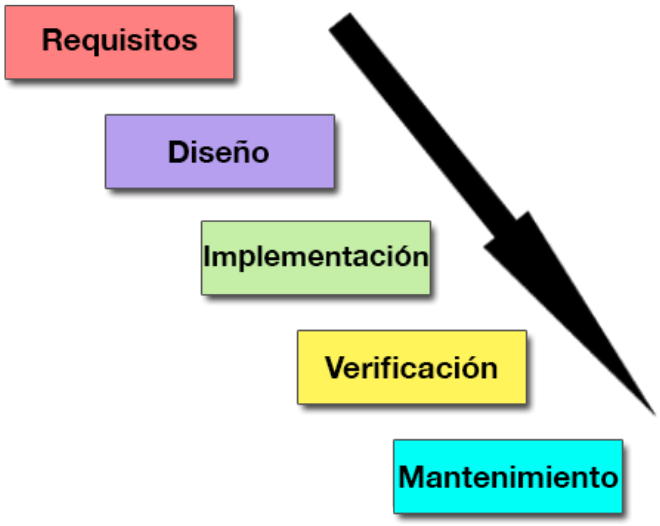

Figura 2.1: Modelo de desarrollo en cascada

Ventajas

La principal ventaja del modelo de desarrollo en cascada es que permite conocer de antemano las etapas por las que pasará el proyecto y permitirá concentrar los esfuerzos del equipo de trabajo en una determinada tarea.

Además cabe destacar que, pese a ser una de las metodologías más criticadas, sigue siendo una de las más utilizadas por la industria del desarrollo de software.

Palabra clave: *Focus*

Desventajas

El modelo de desarrollo en cascada cuenta con una gran desventaja debido a que los cambios en las etapas más avanzadas del proyecto implican mayores esfuerzos y costes del desarrollo[1].

Hay que tener en cuenta que, por esta razón, los proyectos reales no siguen una implementación 100 % lineal, tal y como propone esta metodología.

Palabra clave: Linealidad

[1]De aquí viene la denominación de *cascada* (ver figura 2.1)

2.1.2. Modelo incremental

El modelo incremental (también conocido como *Modelo de desarrollo iterativo y creciente*) nació como una alternativa al modelo de desarrollo en cascada para desarrollar software de manera iterativa, aprendiendo de los errores cometidos en anteriores etapas del proceso y volviendo a repetir todas las etapas dentro de una iteración, permitiendo obtener un entregable funcional del sistema al final de la misma.

Dentro de este modelo, al contrario de lo que ocurre en el modelo de desarrollo en cascada, se realizan cambios en el diseño en cada iteración, añadiendo así nuevas características al sistema.

A pesar de ser un proceso iterativo, éste se compone de dos etapas diferenciadas:

1. **Etapa de Inicialización**

   En esta fase se crea una nueva versión del sistema, consiguiendo una versión con la que el usuario pueda interactuar para realimentar el proceso con nueva información. En esta estapa se crea una *lista de control* del proyecto donde se marcan los objetivos de la iteración en curso y que se revisa periódicamente durante el análisis realizado en esta etapa.

2. **Etapa de Iteración**

   Durante esta etapa se realizan los cambios necesarios al sistema basándose en la *lista de control* generada en la etapa anterior. Estos cambios no son únicamente de funcionalidad del sistema, sino también cambios en los requisitos y en el diseño del producto que pueden provocar cambios en la *lista de control* en la siguiente iteración.

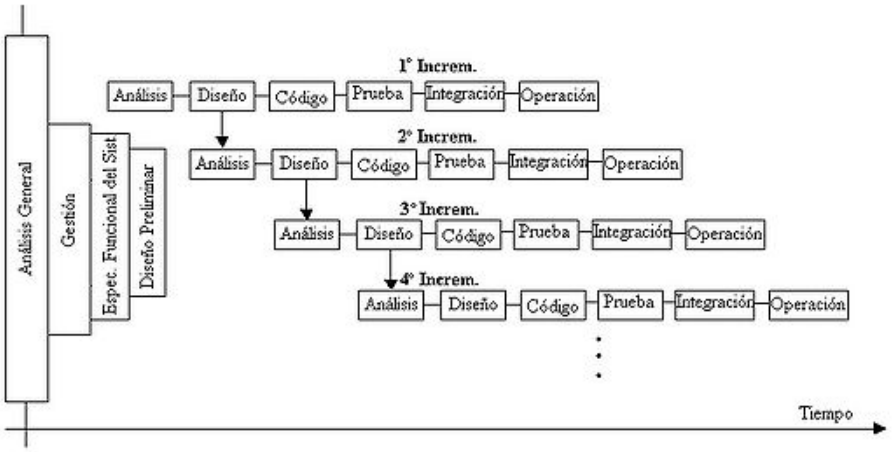

Figura 2.2: Modelo de desarrollo incremental

Ventajas

La principal ventaja del modelo incremental es, sin lugar a dudas, la posibilidad de mejora y rediseño del sistema a lo largo de su fase de desarrollo sin mermar cuantitativamente los plazos de entrega ni aumentar considerablemente los costes de desarrollo.

Esta metodología también proporciona un mayor *feedback* del usuario del sistema durante su desarrollo, lo que aporta comentarios muy útiles para lograr una experiencia del usuario más satisfactoria.

Palabra clave: Iteraciones

Desventajas

Esta metodología tiene una gran desventaja que puede llegar a ser un problema en algunos proyectos, dado que requiere de un cliente involucrado en todo el proceso de desarrollo. No todos los clientes están dispuestos a realizar este esfuerzo, a pesar de que puede suponer un gran beneficio obteniendo un producto final más refinado respecto a sus gustos.

Palabra clave: Involucramiento

2.1.3. Modelo de prototipado

El modelo de prototipado, o más comúnmente conocido como *modelo de prototipos*, es una de las prácticas de desarrollo de software evolutivo.

Esta práctica basa su filosofía en el desarrollo de prototipos del sistema que se quiere construir, generándolos de forma rápida y utilizando la menor cantidad de recursos para que se generen distintos prototipos en el menor tiempo y coste posibles.

El modelo de prototipado suele comenzar realizando prototipos de bajo nivel, construidos con lápiz y papel (o soporte informático que otorgue un resultado similar), donde lo que importa es la ubicación de los elementos en pantalla y la funcionalidad del sistema, más que la apariencia visual, ya que se realizan sin colores ni otros añadidos que limiten la capacidad de concentración del usuario para desarrollar la tarea encomendada. Estos prototipos son utilizados también en los análisis de usabilidad que se realizan con los usuarios para determinar la viabilidad de los proyectos.

Esta práctica se compone de cinco etapas sobre las que se va iterando hasta desarrollar el sistema definitivo:

1. **Plan rápido**

   Durante esta primera etapa se realiza un breve análisis de los requisitos del sistema para definir el plan de actuación y poder tomar decisiones sobre el prototipo que se va a generar.

2. **Modelado, diseño rápido**

   En la etapa de modelado se realiza un diseño rápido de cómo se va a construir el prototipo y qué características del sistema va a cubrir.

3. **Construcción del prototipo**

   La construcción del prototipo se realiza normalmente en papel para ahorrar costes y tiempo de desarrollo, si bien se pueden utilizar otros materiales que sean más fieles al aspecto final del sistema que se va a desarrollar.

4. **Desarrollo, entrega y retroalimentación**

   En base al prototipo construido, el usuario puede realizar pruebas sobre el mismo para determinar los posibles errores de concepto que el ingeniero de software haya podido cometer.

5. **Comunicación**

   Mediante la constante comunicación entre el usuario final y el desarrollador, se consigue realizar un refinamiento de los requisitos del sistema de manera progresiva para que el desarrollo sea lo más fluido posible.

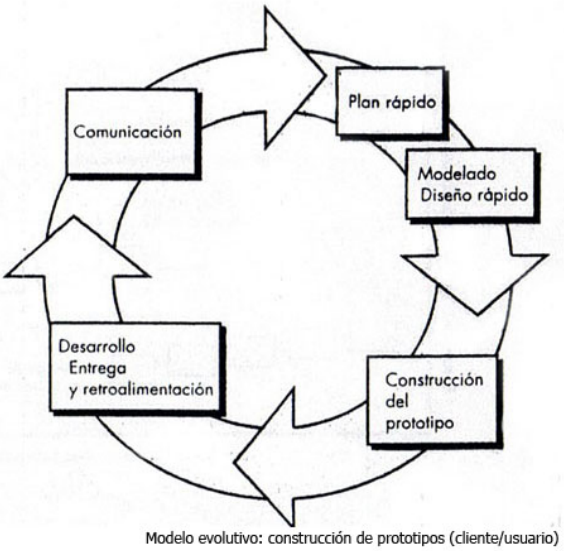

Figura 2.3: Modelo de desarrollo en prototipado

Ventajas

El modelo de prototipado tiene una serie de ventajas respecto al resto de prácticas:

- Reduce el riesgo de construir productos que no cumplan con los requisitos marcados por el cliente, por lo que la satisfacción global del mismo será mayor.

- Reduce el coste de desarrollo debido a la rapidez con la que se alcanza un consenso en los requisitos y el desarrollo en papel hace que el coste del prototipo sea muy bajo.
- El desarrollador se centra en la funcionalidad y diseño de la plataforma sin dejar de lado aspectos importantes como el análisis de los requisitos y, de paso, el prototipo le permite realizar análisis de usabilidad.

  Palabra clave: Pragmático

Desventajas

El modelo de prototipado cuenta con una serie de desventajas que tienen que ser bien gestionadas por parte del desarrollador, ya que se puede llegar a poner en peligro la viabilidad del proyecto:

- El usuario tiende a generar falsas expectativas sobre el sistema final cuando ve por primera vez un prototipo y puede desilusionarse generando un sentimiento enfrentado por pensar que el sistema aún no ha sido construido, pudiendo llegar a ver al equipo de desarrollo como *irresponsables que juegan con su trabajo*.
- El desarrollador, a su vez, puede caer en la tentación de ampliar el prototipo o de elegir incorrectamente un diseño del sistema por realizar los prototipos con rapidez, pudiendo dejar de lado los compromisos de calidad y mantenimiento que el cliente exige.

  Palabra clave: Precipitación

2.1.4. Modelo por etapas

El modelo de desarrollo por etapas es una práctica similar a la del modelo de prototipado ya que muestra al usuario el sistema en distintos estados sucesivos del desarrollo.

La gran diferencia con el modelo de prototipado es que el modelo por etapas no tiene unos requisitos y unas necesidades especificadas desde el principio, sino que se van desarrollando con ayuda del usuario a lo largo de todo el proceso.

El modelo de desarrollo por etapas define las siguientes fases que se van repitiendo consecutivamente en cada etapa del diseño:

1. **Especificación conceptual**

   Se realiza junto al cliente una especificación del problema que se quiere abordar para obtener la mejor solución al mismo.

2. **Análisis de requisitos**

   Se analizan los requisitos que requiere el sistema para que el producto final cumpla con los objetivos iniciales, dentro de esta etapa del desarrollo.

3. **Diseño inicial**

   En esta etapa se realiza un diseño inicial del sistema que se discute con el usuario antes de pasar a la siguiente fase, por si hubiera que depurar alguna de las decisiones tomadas en las fases anteriores.

4. **Diseño detallado, codificación, depuración y liberación**

   La última etapa de la fase incluye el diseño detallado, la codificación, pruebas unitarias y de sistema, y el lanzamiento de una nueva característica del sistema, que puede pasar a producción.

Ventajas

El modelo por etapas proporciona mucho *feedback* por parte del cliente, lo que proporciona que la satisfacción global sea mayor.

Además, el usuario se siente parte del desarrollo y es capaz de seguir la evolución del producto a lo largo de todas las fases del proyecto, aportando mejoras en cada uno de los pasos.

Palabra clave: Estratificación

Desventajas

Esta práctica requiere un alto grado de involucramiento por parte del usuario, lo que en ocasiones no es posible.

La definición de las especificaciones del producto a lo largo de su desarrollo, pese a resultar positiva por la generación sucesiva de necesidades del sistema, puede provocar un encarecimiento del producto por implementar más funcionalidades de las necesarias, o el olvido de alguna funcionalidad que implique posteriormente cambios en el diseño en etapas más avanzadas del desarrollo.

Palabra clave: Indeterminación

2.1.5. Modelo en espiral

El modelo de desarrollo en espiral fue definido por primera vez por Barry Boehm en 1988, siendo también es conocido como *modelo evolutivo en espiral* [Boe88].

La mayor característica de esta práctica de desarrollo es que tiene en cuenta en todo momento el riesgo que existe a la hora de desarrollar un producto software. Por ello, se analizan las distintas alternativas optando por la más asumible y se realizan ciclos sobre la espiral.

Estos ciclos son realizados iterativamente hasta que el producto no necesita ningún refinamiento extra, procediendo a realizar una etapa de puesta en marcha del producto.

Comúnmente se entiende que dentro de cada ciclo de la espiral el proceso de desarrollo sigue una metodología de desarrollo en cascada, aunque puede no ser así.

En cada ciclo de la espiral se tienen en cuenta los objetivos que se deben cumplir al final del mismo y las distintas alternativas de desarrollo, para poder optar por la que sea más viable.

El nombre de modelo en espiral viene dado por su forma de representar el flujo del desarrollo del proyecto en forma de caracola sobre un eje de coordenadas bidimensional (ver figura 2.4). El proyecto, al igual que la caracola, comienza por el anillo interior que se sitúa en el eje de coordenadas y define dos dimensiones:

- **Angular:** Indica el avance del proyecto dentro de un ciclo.
- **Radial:** Indica el coste del proyecto. Cuanto mayor sea el radio de la espiral más caro resultará el desarrollo ya que cada nuevo ciclo que se realiza aumenta el tiempo de desarrollo.

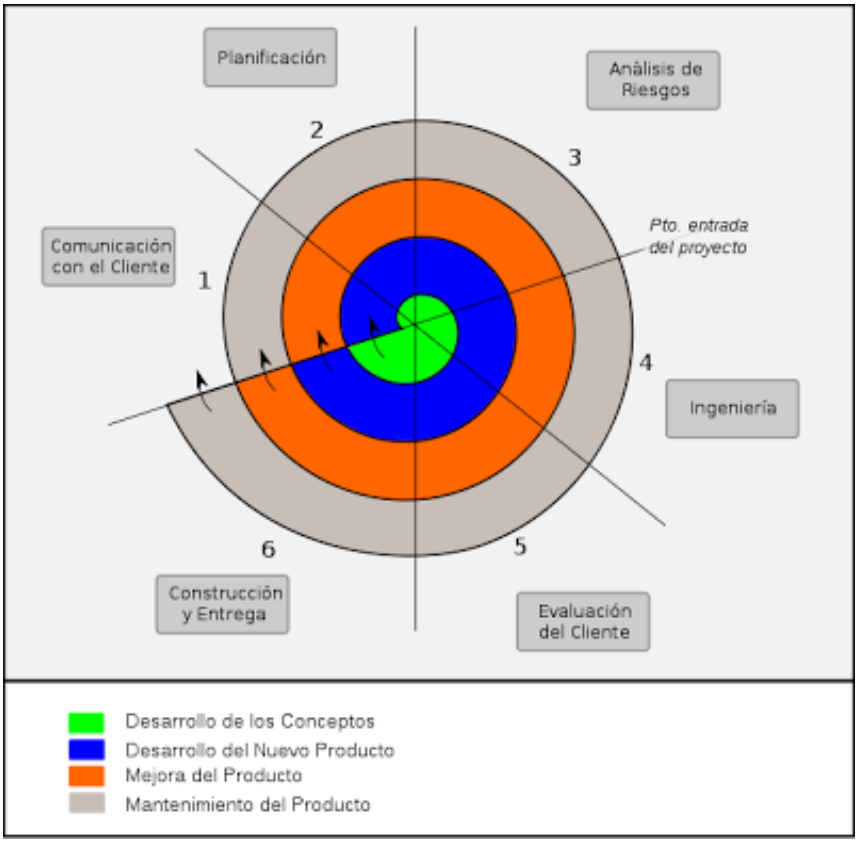

Figura 2.4: Modelo de desarrollo en espiral

Dentro de cada ciclo se realizan cuatro actividades o etapas tal y como se observa en la figura 2.5.

1. **Determinar objetivos**

   Se determinan los objetivos del ciclo, así como los productos que se van a obtener al final del mismo. También se identifican los posibles riesgos en el desarrollo dentro de este ciclo. El primer ciclo añade una planificación inicial del proyecto para poder estimar la duración y los costes del mismo.

2. **Análisis del riesgo**

   Durante esta etapa se analizan en profundidad las posibles amenazas y eventos no deseados así como qué consecuencias pueden llegar a producir.

3. **Desarrollar y probar**

   Se desarrolla la parte del sistema especificada analizando las alternativas sugeridas en la etapa anterior e identificando la resolución de los riesgos. Además, se elige un modelo para el desarrollo, que puede ser cualquiera de los otros existentes, basándose en la alternativa más apropiada.

4. **Planificación**

   En la etapa final se evalúa todo el trabajo realizado decidiendo si se continúa con más ciclos y definiendo el plan de ataque para el próximo ciclo.

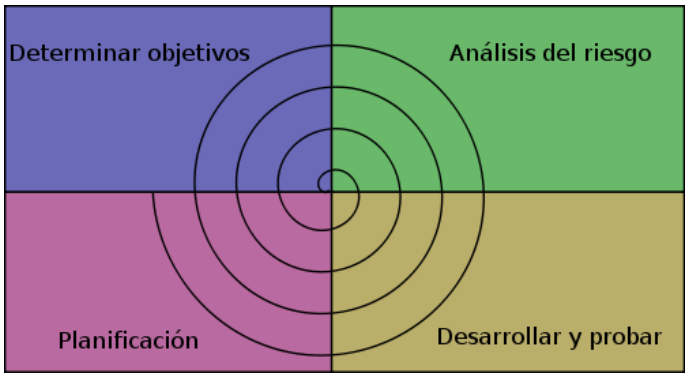

Figura 2.5: Etapas del modelo de desarrollo en espiral

Ventajas

El modelo de desarrollo en espiral posee una serie de ventajas claramente asociadas a su modelo de definición del riesgo existente en el desarrollo de productos software. Estas ventajas son:

- Se reducen riesgos del proyecto, con lo que se minimiza la probabilidad de fracaso asociada a todo proyecto.

- Se introducen objetivos de calidad al evaluar, en todo momento, las mejores alternativas de diseño frente a los distintos riesgos del proyecto.
- Se integra el desarrollo con el mantenimiento, lo que hace más sencillo incorporar nuevas características a funcionalidades ya generadas.

  Palabra clave: Control

Desventajas

Las desventajas del desarrollo en espiral vienen asociadas al modelo cíclico que puede alargar el proyecto más de la cuenta, haciéndolo a su vez más costoso. Estas desventajas son:

- Mayor tiempo de desarrollo del sistema.
- Mayores costes asociados a toda la evaluación de los riesgos e identificación de posibles problemas.
- Necesidad de ingenieros altamente cualificados para una correcta identificación de los riesgos.

  Palabra clave: Coste

2.1.6. Método en V

El método en V (también conocido como *Método-V*) es una metodología estándar utilizada por la Administración Federal Alemana y por el Ministerio de Defensa alemán. Su especificación es pública, por lo que muchas compañías lo han adaptado dentro de sus procesos de desarrollo de software.

Este método se denomina así por su representación gráfica en forma de V (ver figura 2.6), que representa la evolución del desarrollo de un producto. La parte izquierda de la V, en bajada, representa la creación de las especificaciones del sistema y el análisis de los requisitos. Por otro lado, la parte derecha de la V, en subida, representa la verificación y validación. La parte inferior, donde se encuentran ambas partes, representa el período de implementación.

Este método es muy similar al ciclo de desarrollo en cascada dado que posee unas pautas muy rígidas.

El método en V define las siguientes fases:

- **Definición del proyecto**
  - **Conceptos de operaciones**
    Definición del sistema sin entrar en detalle.

- **Requisitos y arquitectura del sistema**
  Se definen los requisitos y el diseño de alto nivel del sistema.
- **Diseño detallado**
  Antes de llegar a la fase de implementación, se realiza el diseño detallado de los componentes que generarán el sistema.

- **Implementación**
- **Pruebas e integración**

  - **Integración, pruebas y verificación unitarias**
    Se realizan pruebas unitarias y de verificación e integración de los módulos y submódulos de los que se compone el sistema.
  - **Verificación y validación del sistema**
    Se realizan las pruebas de integración y validación del sistema completo.
  - **Mantenimiento**
    Para finalizar, el proceso se queda en fase de mantenimiento donde se realizarán las modificaciones necesarias del sistema (detección y corrección de errores).

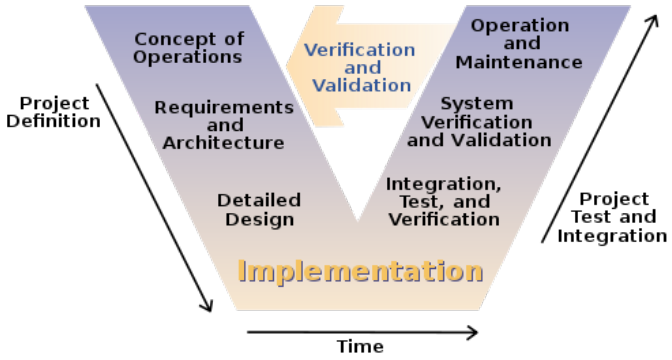

Figura 2.6: Método de desarrollo en V

Ventajas

El método en V comparte casi todas las ventajas que ofrece el ciclo del desarrollo en cascada.

Como ya se comentó anteriormente, la principal ventaja es el conocimiento de antemano de las etapas por las que pasará el proyecto y que podrá permitir que el grupo de desarrollo concentre sus esfuerzos en una determinada tarea, disminuyendo los riesgos en la realización de ésta.

Palabra clave: *Focus*

Desventajas

Como es de esperar, este modelo, pese a ser más completo y algo más flexible al determinar una mayor cantidad de fases de desarrollo, posee las mismas desventajas que el modelo de desarrollo en cascada.

Al generarse las pruebas después de la implementación y debido a la rigidez del sistema, puede efectuarse un proceso de vuelta a una fase anterior contraria a la naturaleza del propio método, pero que normalmente es necesaria en el desarrollo de los productos software.

Palabra clave: Rigidez

2.1.7. Proceso Unificado

El Proceso Unificado es un proceso iterativo e incremental, dirigido por casos de uso y centrado en la arquitectura. El Proceso Unificado más documentado es el *Proceso Unificado de Rational™ o RUP*[2] de IBM.

Este proceso define cuatro fases:

1. **Inicio**

   La fase de inicio identifica los requisitos y los objetivos del sistema que se va a construir.

2. **Elaboración**

   Se elabora el diseño de la plataforma y se identifican los casos de uso que dirigirán el desarrollo.

3. **Construcción**

   En esta fase se centra el trabajo de desarrollo e implementación dentro de la etapa en la que estamos del proceso.

4. **Transición**

   La transición de una etapa a otra permite identificar el estado del proyecto y continuar dentro de la siguiente iteración del proceso.

Cada una de estas cuatro grandes fases se divide en una serie de disciplinas similares a las definidas en el modelo de desarrollo en cascada. Casi todas las fases incluyen tareas de trabajo en cada una de las iteraciones, pero el grado de esfuerzo depende de la fase en la que se encuentra el proceso. Estas iteraciones son:

[2]del inglés Rational Unified Process, RUP

- Análisis de requisitos
- Diseño
- Implementación
- Prueba

Este proceso está basado en los casos de uso que se utilizan para definir los requisitos funcionales y los contenidos de cada una de las iteraciones que se van a realizar.

El proceso unificado asume la existencia de diversos modelos que definen la arquitectura del sistema software que se está desarrollando. Por ello, es un proceso centrado en los aspectos relacionados con la arquitectura del sistema.

Por otra parte, es un proceso que también se centra en los riesgos, al igual que el modelo en espiral definido en la sección 2.1.5. De esta forma, los mayores riesgos son considerados primero para evitar posibles fallos en el desarrollo del sistema.

Proceso Unificado de Rational

El Proceso Unificado de Rational™ es una adaptación del Proceso Unificado con una serie de pasos definidos no definitivos, sino que se refiere a un conjunto de prácticas que se adaptan dependiendo de las necesidades de cada organización.

El Proceso Unificado de Rational™ se basa en los siguientes seis principios:

1. **Adaptación del proceso**

   Todo el proceso debe adaptarse a las necesidades del cliente y se estima necesaria una interacción con el mismo. Dependiendo del proyecto y de las características intrínsecas de la organización que lo realiza, puede ser implementado de una forma u otra.

2. **Equilibrio de prioridades**

   Se realiza una negociación de los requisitos entre todas las partes involucradas llegando a un acuerdo que deben cumplir y que satisfaga a todas las partes.

3. **Valor iterativo**

   Los proyectos que siguen este proceso se realizan en fases iterativas que permiten al cliente ver el producto en distintas etapas del desarrollo, haciendo crecer su valor y calidad gracias a las opiniones recibidas.

4. **Colaboración entre equipos**

   El desarrollo del software se realiza entre varios equipos de desarrollo que deben tener una comunicación fluida para que el proyecto sigua el rumbo determinado.

5. **Aumento del nivel de abstracción**

   Al aumentar el nivel de abstracción se consigue que los equipos de desarrollo se centren más en la consecución del producto en base a los productos y no en los detalles más concretos de su implementación, aportando así diferentes soluciones en distintas etapas y versiones arquitectónicas sustancialmente distintas pero coherentes para presentar la mejor solución posible.

6. **Enfoque en la calidad**

   El control de la calidad del desarrollo del producto debe realizarse en todos y cada uno de los aspectos de la producción y no sólo al alcanzar ciertos hitos.

El Proceso Unificado de Rational™ define dos grandes fases divididas en varias etapas, tal y como se puede observar en la figura 2.7.

- **Proceso**
  - Modelado de negocio
  - Requisitos
  - Análisis y diseño
  - Implementación
  - Pruebas
  - Despliegue
- **Soporte**
  - Gestión del cambio y configuraciones
  - Gestión del proyecto
  - Entorno

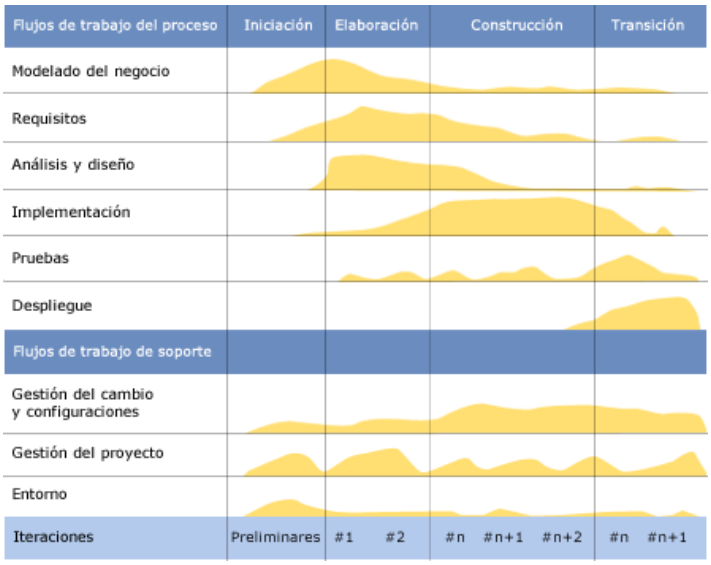

Figura 2.7: Proceso de desarrollo unificado

Al ser un proceso dinámico que se adapta a las distintas necesidades de los proyectos y las organizaciones, cumple con las cuatro fases en la sección 2.1.7.

Ventajas

Las principales ventajas en el uso del Proceso Unificado de Rational™ radican en su adaptabilidad a los distintos escenarios de uso donde se aplique.

Esto favorece una rápida reacción ante los posibles riesgos que pueda tener el proyecto.

Gracias a la retroalimentación surgida por ser un proceso iterativo, hace que el producto se ajuste a las necesidades reales del cliente.

Palabra clave: *Adaptabilidad*

Desventajas

Este proceso tiene un alto grado de complejidad que puede resultar no adecuado para todo tipo de proyectos. De hecho, es poco recomendable su uso en proyectos de corta duración.

Además, al ser un método complejo requiere de ingenieros bien formados en el proceso.

Palabra clave: *Complejidad*

2.2. Prácticas de desarrollo ágil

A lo largo de esta sección se muestran los distintos tipos de prácticas de desarrollo ágil que han ido evolucionando desde finales de los años 80 hasta su masiva incorporación a la gestión de productos software durante la primera década del siglo XXI.

Sin embargo, pese a existir anteriormente algunas de estas prácticas, no fue hasta febrero de 2001 cuando 17 desarrolladores sentaron las bases de las prácticas ágiles proclamando lo que se conoce como el *Manifesto for Agile Software Development* [Agi13] y que todas las prácticas ágiles aceptan en sus propuestas. El *Agile Manifesto* establece cuatro valores clave:

- **Las personas y sus interacciones** se valoran sobre los procesos y las herramientas.
- **El software que funciona** se valora sobre la documentación extensa.
- **La colaboración del cliente** se valora sobre la negociación del contrato.
- **La respuesta al cambio** se valora sobre el seguimiento de un plan concreto.

 **Nota.-** Aunque el *Agile Manifesto* valora más los términos resaltados en negrita, esto no interfiere con la necesidad de la existencia de los elementos a la derecha.

Así mismo, el *Agile Manifesto* proclama doce principios en los que se basa:

1. La mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2. Se acepta que los requisitos cambien, incluso en etapas tardías de desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar una ventaja competitiva al cliente.
3. Se entrega software funcional con cierta frecuencia, entre dos semanas y dos meses, con preferencia al período de tiempo más corto posible.
4. Los responsables del negocio y los desarrolladores trabajan juntos de forma cotidiana durante todo el proyecto.
5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo necesario, confiándoles la ejecución del trabajo.
6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros son las conversaciones cara a cara.
7. El software funcionando es la principal medida de progreso.
8. Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios deben ser capaces de mantener un ritmo constante de forma indefinida.

9. La atención continua hacia la excelencia técnica y el buen diseño mejora la Agilidad.

10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.

11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.

12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para, a continuación, ajustar y perfeccionar su comportamiento en consecuencia.

Basándose en estos principios, se enumeran a continuación algunas de las prácticas ágiles más conocidas.

2.2.1. Scrum

Scrum es un método de desarrollo de productos software, basado en un proceso iterativo e incremental, en el que los productos se desarrollan en pequeñas piezas que se van construyendo sobre las piezas creadas anteriormente.

La construcción de los productos pieza por pieza favorece la creatividad y permite que el desarrollo se realice de una forma más ágil, siendo una práctica más eficaz contra posibles imprevistos y construyendo únicamente aquello que se necesita, dada la evaluación constante de los requisitos del producto.

Scrum es, en realidad, un conjunto de procesos que definen una serie de prácticas y roles, que interaccionan entre sí a lo largo de las distintas iteraciones, denominadas *sprints*.

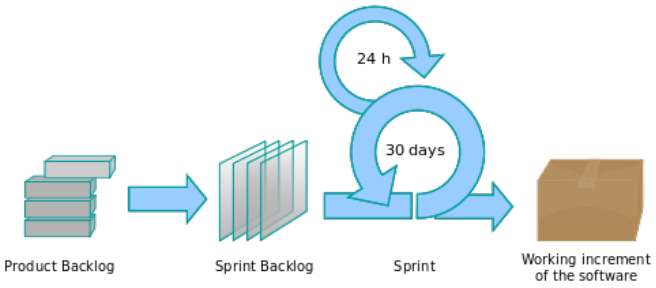

Figura 2.8: Scrum y sus sprints

Un *sprint* tiene una duración definida por el equipo de desarrollo de entre 1 y 4 semanas, siendo la práctica más extendida el uso de *sprints* de 4 semanas. Al final de cada *sprint* el equipo de desarrollo debe tener una versión incremental del producto que sea potencialmente entregable, es decir, debe haber generado un prototipo completamente

funcional de las características definidas para dicho *sprint*, aunque normalmente no sea un desarrollo que use el usuario final, pero que sirve para seguir avanzando en tiempo y forma en el proyecto.

Esta práctica está bastante ligada a las historias de usuario, que son representaciones limitadas de los requisitos pero utilizando un lenguaje común del usuario y que deben ser cuantificables para que su esfuerzo de desarrollo pueda ser estimado. Se verán con más detenimiento en la sección 2.3.1.

Utilidades en Scrum

Dentro del método Scrum, existen una serie de herramientas o utilidades que sirven de referencia al equipo y sobre las que se realizan diversas acciones a través de las distintas reuniones planificadas.

A continuación, se pueden ver las distintas utilidades que propone Scrum:.

- **Product Backlog**

  Es una lista ordenada de "requisitos" de un producto. Contiene una serie de elementos que son ordenados por el *Product Owner* (ver sección 2.2.1) teniendo en cuenta parámetros como el riesgo, el valor de negocio, dependencias, fechas de implantación, etc.

  El *Product Backlog* es lo que se va a construir ordenado de la forma en la que debe ser construido. Es editable por cualquier miembro del equipo, pero el *Product Owner* es el último responsable de su orden antes de que llegue al equipo de desarrollo.

  A pesar de que el *Product Backlog* es responsabilidad del *Product Owner*, el esfuerzo estimado para completar cada una de las historias es determinado por el equipo de desarrollo. El equipo contribuye estimando las tareas y las historias de usuario en horas o en puntos de esfuerzo.

- **Sprint Backlog**

  Es la lista de trabajo que debe manejar el equipo de desarrollo durante el siguiente *sprint*. Esta lista se obtiene seleccionando historias y tareas de la parte superior del *Product Backlog* hasta que el equipo estima que son suficientes para rellenar el *sprint*.

  El equipo de desarrollo debería tener en cuenta la velocidad de desarrollo obtenida en *sprints* previos a la hora de seleccionar las tareas de un nuevo *sprint*.

Las historias se dividen en tareas más pequeñas por el equipo de desarrollo que suelen tener entre 4 y 16 horas de trabajo para que cualquier miembro del equipo pueda coger una tarea y realizarla en, a lo sumo, dos días de trabajo. Las tareas del *Sprint Backlog* nunca son asignadas, sino que son seleccionadas por los miembros del equipo durante la reunión diaria (ver sección 2.2.1) teniendo en cuenta su prioridad y las habilidades de los miembros del equipo.

El *Sprint Backlog* es propiedad del equipo de desarrollo y suele ir acompañado de una pizarra de tareas para visualizar y cambiar el estado de las tareas del *sprint*: "Por hacer", "En progreso", "En pruebas" y "Hecho".

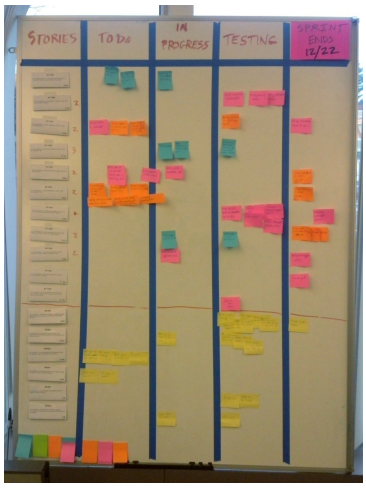

Figura 2.9: Ejemplo de panel de Scrum

- **Increment**

  Es la suma de todos los elementos completados del *Product Backlog* durante un *sprint* y todos los precedentes. En términos comunes, es la versión alfa, beta o demo del producto y debe ser completamente funcional para las características implementadas, independientemente de que el *Product Owner* decida lanzarlo o no.

- **Burn down**

  Es una gráfica visible para todo el equipo que muestra el trabajo restante en el *Sprint Backlog*. Se actualiza cada día dando de un vistazo el estado del *sprint*. Idealmente muestra para la duración del *sprint*, las horas (o el esfuerzo) restantes y el número de tareas restantes y completadas, pudiendo ver si el *sprint* sufrirá algún retraso o no.

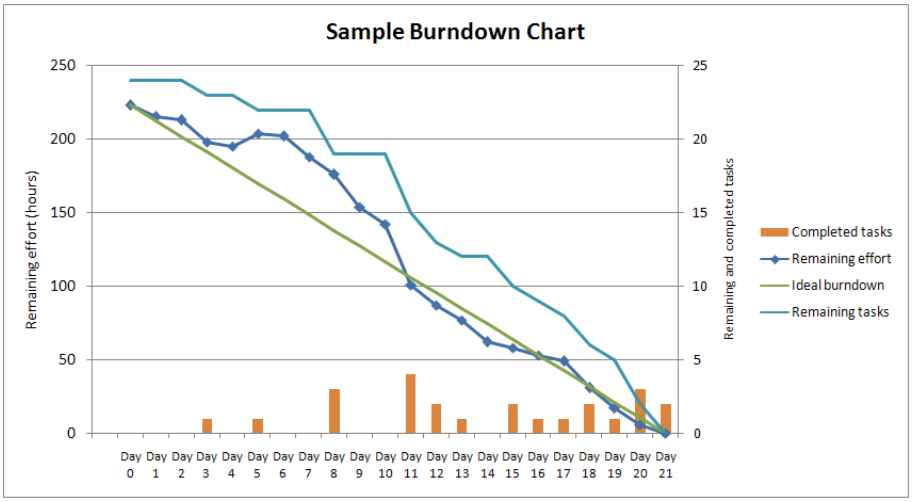

Figura 2.10: Ejemplo de burn down

Roles en Scrum

Como se ha mencionado anteriormente, Scrum define una serie de roles que deben ser asignados a distintas personas para la realización del proyecto. En esta sección se podrán conocer cuáles son los roles involucrados y cuál es su función.

- **Product Owner**

  Representa la voz del cliente dentro del proyecto. Escribe historias de usuario (ver sección 2.3.1), las prioriza y las coloca en el Product Backlog.

- **ScrumMaster o Facilitador**

  Se le denomina facilitador porque su trabajo radica, primordialmente, en eliminar los obstáculos que impidan que el equipo alcance el objetivo del *sprint*. El *ScrumMaster* no es el líder del equipo, que es autosuficiente, sino que se encarga de centrar a los miembros del equipo, resolviendo los posibles problemas que puedan surgir y haciendo que las reglas del Scrum se cumplan.

- **Equipo de desarrollo**

  Es el equipo que desarrolla de manera efectiva el producto. Suele tratarse de equipos de menos de 10 personas con habilidades transversales que permiten la consecución del trabajo.

- **Stakeholders**

  Es un rol auxiliar que representa a los clientes, proveedores, vendedores, etc. Es conveniente involucrar a otros usuarios durante las revisiones del *sprint* para que hagan posible el proyecto, ya que aportan un valor añadido al proceso y son los usuarios a los que el proyecto supondrá un beneficio.

- **Managers o Administradores**

  Es un rol auxiliar que representa a aquellas personas que establecen el ambiente necesario para el desarrollo del producto.

Reuniones en Scrum

El proceso de desarrollo ágil Scrum define una serie de reuniones que se deben seguir:

- **Scrum diario**

  Se realiza una reunión diaria entre todos los roles implicados, si bien todo el mundo involucrado es bienvenido. Estas reuniones se rigen por las siguientes características:

  - Las reuniones diarias comienzan siempre a la misma hora, incluso aunque falte alguno de los miembros.

- Las reuniones deben realizarse en el mismo lugar y tienen una duración prefijada de 15 minutos, que no se podrá alterar, para no entorpecer el trabajo normal de la empresa.
- Los miembros deben responder a las preguntas:
  - ¿Qué hiciste ayer?
  - ¿Qué tienes pensado hacer hoy?
  - ¿Algún problema que dificulte avanzar en dicha tarea?

- **Estimación del Backlog**

  El equipo debe realizar una reunión por cada *sprint* de forma que se estime la duración de una tarea. Esta estimación puede realizarse de manera temporal o por puntos de esfuerzo relativos a las tareas.

- **Scrum de Scrums**

  Cada día, después de realizar la reunión diaria, se puede celebrar una reunión entre los distintos equipos de trabajo que pueden llevar sus propios Scrum internos, para descubrir el avance del desarrollo y discutir los posibles problemas de interacción y necesidades entre los distintos equipos de trabajo. La agenda de un Scrum de Scrums es la misma que la del Scrum diario incluyendo, además, las siguientes preguntas:

  - ¿Qué ha hecho tu equipo desde la última reunión?
  - ¿Qué hará hoy tu equipo?
  - ¿Hay algo que entorpezca el avance de tu equipo?
  - ¿Tu equipo va a hacer algo que pueda entorpecer el avance de otro equipo?

- **Planificación del Scrum**

  Al comienzo de cada *sprint* se debe realizar una reunión de planificación donde se deben abordar los siguientes aspectos:

  - Seleccionar el trabajo que se va a realizar durante el *sprint*.
  - Preparar, junto a todo el equipo, el *backlog* que detalla el tiempo que llevará realizar dicho trabajo.
  - Identificar qué cantidad del trabajo es razonable pensar que estará terminado durante el *sprint*.
  - La reunión deberá tener una duración máxima de 8 horas, en las que idealmente se repartirá el tiempo de la siguiente manera:
    - Primeras 4 horas: Todo el equipo debatirá la prioridad de las tareas en las que se descompone el *backlog*.
    - Últimas 4 horas: El equipo de desarrollo realizará un plan para el *sprint*, obteniendo el *backlog* del *sprint*.

- **Fin de ciclo**

  Al finalizar un *sprint* se debe realizar una reunión de fin de ciclo, que se compone a su vez de dos reuniones denominadas "Revisión del *sprint*" y "Retrospectiva del *sprint*".

  - **Revisión del *sprint***
    - Tiene una duración máxima de 4 horas.
    - Revisión tanto del trabajo completado, como del que no se ha podido completar.
    - Presentación del trabajo terminado a los *stakeholders*.
  - **Retrospectiva del *sprint***
    - Tiene una duración máxima de 3 horas.
    - Intervención de todos los miembros pensando en el *sprint* pasado.
    - Puesta en común de dos preguntas: ¿Qué fue bien durante el *sprint*? y ¿Qué se puede mejorar en el siguiente *sprint*?

Ventajas

Una de las mayores ventajas de Scrum es que permite que el equipo de desarrollo conozca en todo momento el estado del proyecto, pero sin dejar de lado al cliente ya que forma parte del proceso de desarrollo. Asimismo, el cliente es capaz de ver resultados en cortos períodos de tiempo, con lo que aumenta la satisfacción con el producto final.

Los desarrolladores, a su vez, tienen una mayor libertad de trabajo puesto que son ellos mismos quienes eligen las tareas que quieren realizar, con lo que se obtiene un mayor compromiso por terminarlas a tiempo.

Además, el *burn down* permite que todo el mundo involucrado en el proyecto conozca las fechas límite, así como si éstas van a poder ser cumplidas o no.

Palabra clave: *Conocimiento*

Desventajas

La mayor desventaja que posee Scrum es que requiere un cliente involucrado con el proyecto y que esté dispuesto a realizar ciertos sacrificios, generalmente de tiempo, para que todo el proceso funcione como se espera.

También hay que tener en cuenta que requiere un mayor compromiso por parte del equipo de desarrollo y de los roles principales ya que se debe asistir a las reuniones programadas y esto conlleva un alto grado de organización personal para que no se produzcan retrasos.

Palabra clave: *Compromiso*

2.2.2. Kanban

Kanban es una nueva técnica de desarrollo de software formulada por David J. Anderson de manera eficiente para que las funcionalidades a implementar estén desarrolladas lo antes posible sin sobrecargar a los desarrolladores. Esta idea está basada en el sistema de producción de los vehículos Toyota™ denominado *JIT*[3].

Kanban utiliza un sistema de trabajo continuo basado en la limitación inherente que un desarrollador tiene a la hora de analizar, desarrollar y probar las nuevas funcionalidades que requiera un sistema software. Al igual que otras prácticas ágiles se apoya en un panel sobre el que se van colocando las cartas que representan las diferentes historias de usuario que hay que desarrollar. La gran diferencia con otros métodos es que Kanban limita el número de tareas que puede haber en cada columna ayudando de esta forma a identificar mejor los cuellos de botella que se pudieran producir, generando un flujo de trabajo más realista y eficaz.

Esta práctica no fija una serie de pasos a seguir, al igual que tampoco detalla unos roles prefijados, ya que comienza con los roles que hubiese en un principio y se va adaptando y evolucionando a medida que el sistema lo hace de forma incremental. No obstante, sí que fija una serie de prácticas que deben seguirse para realizar un correcto seguimiento de este método de desarrollo.

1. **Visualización**

   Se considera primordial poder visualizar el proyecto para comprender mejor cómo funciona. De esta forma, los desarrolladores pueden entender el flujo de trabajo que se debe seguir para realizar mejoras sobre el sistema. Esto se consigue gracias al uso de un panel con columnas que delimitan los estados de las tareas dentro del flujo de trabajo.

2. **Limitar el trabajo en proceso**

   Cada una de las columnas limita el número de tareas que puede contener. Como ya se ha comentado anteriormente, esto ayuda a identificar las debilidades del proyecto y, a su vez, hace que el sistema de desarrollo funcione a un ritmo constante ya que, cada vez que se termina una tarea en las columnas situadas más a la derecha, se genera un hueco que es rellenado sucesivamente, moviendo de este modo las tareas de las columnas que están más a la izquierda hacia la derecha, sin superar el límite por columna establecido.

3. **Controlar el flujo de trabajo**

   El paso de una tarea entre las distintas fases del flujo de trabajo debe ser monitorizado y cuantificado para que, de una forma evolutiva, se puedan evaluar los impactos positivos o negativos que se están transmitiendo al desarrollo del producto.

[3]del inglés *Just-In-Time* - TDD

4. **Hacer pública la forma de trabajo**

   Cada miembro del equipo o cada parte del equipo encargada de una tarea, debe hacer pública de la mejor manera posible su forma de trabajo para que todo el equipo de desarrollo pueda conocer mejor cómo funcionan internamente y, de esta forma, realizar un análisis más racional, empírico y objetivo con el que conseguir debatir y proponer las mejores sugerencias de mejora.

5. **Implementar métricas de mejora**

   La mejor forma de progresar es mediante la colaboración de los distintos equipos de trabajo aportando métricas que puedan otorgar una mayor interacción que haga que el equipo evolucione como un todo.

6. **Mejorar colaborando, evolucionar experimentando**

   Si el equipo colabora y conoce mejor el trabajo y el flujo que debe seguir el proceso de desarrollo y los posibles riesgos, se pueden obtener mejores ideas de progreso y llegar a soluciones mediante un consenso general. No obstante, es el propio método el que persigue cambios pequeños, incrementales y evolutivos en el desarrollo que sean invariables, y eso se consigue con una evolución en conjunto del equipo.

Ventajas

Kanban es una práctica centrada en la constante mejora del producto, por lo que una de sus ventajas es la calidad del producto resultante debido a una mejor detección de los defectos que se puedan haber insertado en las etapas de desarrollo, ya que se reduce el número de tareas con las que el equipo está trabajando y en las que se puede estar más concentrado.

Además, en grandes empresas es interesante la posibilidad de tener un control de la producción o de poder generar una producción flexible en base a la demanda, aunque no pueda parecer, a priori, aplicable a todos los casos de la Ingeniería Informática.

Palabra clave: *Calidad*

Desventajas

En sistemas software en los que las tareas del equipo se vean ralentizadas por agentes externos de los que depende la producción, el uso de esta práctica no está recomendado ya que el equipo estaría gran parte del tiempo desocupado.

Además, esta técnica requiere un esfuerzo muy grande de comunicación entre todas las partes involucradas para que el proceso de desarrollo se vea afectado lo menos posible.

Palabra clave: *Comunicación*

2.2.3. Extreme programming

Extreme programming (XP) es una práctica ágil de desarrollo de software que busca como objetivo principal responder a las necesidades de los clientes, centrándose a su vez en los cambios que los requisitos suelen sufrir a lo largo del desarrollo de un proyecto.

Fue introducida a finales de los años 90, con uno de sus primeros "casos de éxito" logrados por Kent Beck, quien la utilizó en el proyecto C3 (Chrysler Comprehensive Compensation) para la firma Chrysler.

El modelo de funcionamiento de XP se basa en el hecho de la existencia de cuatro variables presentes en todo proyecto software: *coste*, *tiempo*, *calidad* y *alcance*. Tres de estas cuatro variables son fijadas por agentes externos al equipo de desarrollo (clientes, proveedores, distribuidores, jefes de proyecto, etc.), mientras que la variable libre la establece el equipo de desarrollo.

Dado que la mayoría de los proyectos no tienen todos los requisitos al comienzo del mismo, XP propone un ciclo de vida del proyecto dinámico que permite adaptarse a esta característica intrínseca del desarrollo de software.

Esta adaptación al entorno cambiante se consigue mediante iteraciones con sistemas con características funcionales al final del mismo, donde se realizan, dentro de cada iteración, ciclos completos de análisis, diseño, implementación y pruebas.

XP define un conjunto de fases que se detallan a continuación:

- **Planificación**

  Durante esta fase se recopilan las historias de usuario que definirán el producto para que los desarrolladores puedan hacer una estimación, la cual afectará al Plan de Entregas que se desarrollará durante esta fase y al Plan de Iteraciones donde se definirán las fechas de las entregas parciales.

- **Diseño**

  Durante esta fase el equipo de desarrollo realiza diseños simples y claros respecto a la iteración actual sobre la que se esté trabajando. En caso de encontrar algún problema técnico o si no se puede estimar el tiempo necesario para implementar una determinada historia de usuario, se propone el uso de soluciones *spike*, que consisten en pequeñas soluciones que ayudan a dicha evaluación. Durante esta fase también se suele realizar un *refactoring* para hacer más eficiente y legible el código inicial.

- **Desarrollo**

  Es requisito indispensable que el cliente esté disponible durante todo el proyecto, ya que proporciona no solo las historias de usuario, sino también los detalles concretos que se necesitan en el diseño de bajo nivel o diseño detallado. Durante esta

fase XP fomenta el uso de prácticas que serán explicadas posteriormente como: *Pair programming* (ver sección 2.2.4), *test-driven development* (ver sección 2.2.5) e *integración continua* (ver sección 2.2.6).

- **Pruebas**

  Durante esta fase el código se comprueba realizando pruebas unitarias sobre los distintos componentes que conforman la aplicación, que idealmente han sido desarrollados siguiendo la práctica *TDD*, y realizando pruebas de integración que se van verificando gracias al uso de la *Integración continua*.

Ventajas

Una de las mayores ventajas del uso de XP es la libertad que tiene el equipo de desarrollo para poder fijar uno de los cuatro valores establecidos, lo que le otorga cierta flexibilidad a la hora de afrontar el proyecto.

También hay que tener en cuenta que el grado de satisfación del cliente será mayor ya que puede ir viendo resultados en cortos períodos de tiempo y sintiéndose más realizado al formar parte del desarrollo de su propio producto.

Palabra clave: *Flexibilidad*

Desventajas

Al igual que otras prácticas ágiles, XP requiere un gran compromiso por parte del cliente que debe ser una figura más del equipo de desarrollo, lo cual no siempre es posible y requiere un gran compromiso del cliente que tiene que descuidar otras de sus obligaciones si el proyecto lo requiere.

Desde el punto de vista del desarrollo, centrarse en el diseño únicamente de las historias de usuario que se van a implementar durante una iteración, es una gran desventaja ya que puede provocar cambios posteriores en el diseño si no se tuvieron en cuenta decisiones que afectarían en posteriores iteraciones, lo cual iría en contra de la propia propuesta.

Palabra clave: *Concentración*

2.2.4. Pair programming

Pair programming[4] es una técnica de desarrollo ágil en la que dos programadores trabajan sobre un mismo ordenador.

[4]en español: Programación por parejas

Uno de los programadores es conocido como el "conductor" y es el que escribe el código y maneja tanto el teclado como el ratón. El otro programador es conocido como el "observador" y es el que revisa el código según lo escribe su compañero, en busca de erratas y fallos, además de pensar si el código está avanzando en la dirección correcta para resolver el problema propuesto.

Gracias a esta técnica, el "conductor" se libera de la presión sobre el problema ya que cuenta con el apoyo que le brinda su compañero en caso de cometer algún fallo.

Esta técnica de desarrollo generalmente termina produciendo programas más cortos, con un mejor diseño y un menor número de *bugs*. Además, dos programadores, aunque trabajen en conjunto, suelen terminar la tarea asignada más rápido que si sólo uno de ellos se dedicara a la misma.

Cada cierto tiempo, ambos programadores intercambian el rol para que no se acostumbren a una actitud pasiva ante el problema. Además, en muchas organizaciones se utiliza la técnica conocida como "Pair programming promiscuo" en la que las parejas se van permutando para que el intercambio de conocimientos sea lo más enriquecedor posible.

Teddy bear pair programming

El *Teddy bear pair programming* es una variante del pair programming en la que el "observador" no es otra persona, sino un peluche u objeto inanimado (típicamente un osito de peluche, de ahí su nombre en inglés).

Poco se sabe del origen de esta variante, pero existen referencias que confirman que Universidades como el MIT (Massachusets Institute of Technology) poseían un peluche en sus laboratorios para que los alumnos pudieran utilizarlo como compañero antes de preguntarle la duda al profesor.

Esta variante se basa en el mismo principio que la práctica original, siendo la mejor técnica para comprender el código y el problema el explicárselo detalladamente a alguien (o algo) para que el propio programador se dé cuenta por sí mismo de dónde está el error y cuál es la posible solución al mismo.

Ventajas

El *pair programming* es una técnica que se desarrolla con un par de ventajas muy claras:

- La primera de ellas es la velocidad con la que se resuelven problemas, incluso aquellos más complejos, gracias a la colaboración entre los programadores, dando incluso al código generado una mayor efectividad y legibilidad.

- Por otra parte, la integración de nuevas personas a los equipos de trabajo y el intercambio de conocimientos mediante el uso de esta técnica es un gran factor a tener en cuenta para utilizar esta práctica.

Palabra clave: *Eficiencia*

Desventajas

El *pair programming* requiere de espacio para las dos personas y de un lugar de trabajo en el que se pueda conversar y en el que no haya demasiadas interrupciones ya que la interacción entre los programadores es necesaria e inevitable.

También se puede tener en cuenta que, si ambos programadores son poco experimentados, los resultados obtenidos pueden ser mejores que individualmente pero serán notablemente mejorables.

Palabra clave: *Interrupciones*

2.2.5. Test-driven development

Test-driven development[5], es una práctica de desarrollo ágil que, como su propio nombre indica, centra todo el proceso en el desarrollo de los *tests* que debe pasar el código que se va a generar.

Puede resultar contra-natura pero esta práctica propone generar primero los *tests* de los componentes que se van a desarrollar antes de implementar estos últimos. Como es lógico, estas pruebas fallarán nada más terminarlas puesto que no están implementadas las funciones que harán que éstas ejecuten correctamente. Después, el desarrollador generará el mínimo código necesario para que las pruebas sean satisfactorias para, finalmente, refactorizar el código cumpliendo los estándares oportunos.

Esta técnica ha sido comúnmente relacionada con *extreme programming* (ver sección 2.2.3), pero cada vez gana un mayor número de seguidores por sí misma.

El ciclo de vida del software con esta práctica es el siguiente:

1. El programador (re)escribe un test.
2. Se ejecutan los *tests*. Si todos son correctos, se vuelve al paso 1.
3. Si los *tests* han fallado, se hace un pequeño cambio en el código que se está probando (entre uno y diez cambios máximo). Si no se satisface la prueba rápidamente o si fallan más *tests*, el usuario debe deshacer dichos cambios antes de ponerse a depurar y comenzar de nuevo este paso.

[5]en español: desarrollo centrado en las pruebas

4. Se ejecutan los *tests*. Si fallan, se vuelve al paso 3. Si han ejecutado correctamente, se continúa implementando nuevas pruebas y funcionalidades, volviendo al paso 1.

5. Si se han terminado de implementar todas las funcionalidades y todas las pruebas han ejecutado satisfactoriamente, el desarrollo ha concluido.

Ventajas

El software producido es de mayor calidad, siempre y cuando los *tests* desarrollados sean exhaustivos y comprueben todas las posibles casuísticas.

Esta técnica se puede utilizar junto a cualquier otra práctica ágil y está pensada para generar *tests* de caja blanca, ya que es el propio programador el que genera los *tests* y conoce el código que escribe, aunque se haga a posteriori.

Palabra clave: *Calidad*

Desventajas

El programador debe pensar primero todas las posibles casuísticas del código aun cuando puede no tenerse muy claro cuáles son todos los requisitos del sistema, especialmente en etapas tempranas del desarrollo.

Palabra clave: *Variabilidad*

2.2.6. Integración continua

La integración continua es una práctica que se menciona en combinación a las dos anteriores por ser una de las prácticas propuestas en *extreme programming*.

Esta práctica se basa en la prevención de los problemas de integración que surgen a la hora de combinar los distintos componentes de un sistema software para que interactúen entre ellos.

Normalmente, se suele utilizar esta práctica junto a un control de versiones y a las prácticas del *test-driven development*. Esto asegura que, cuando un miembro del equipo implementa alguna funcionalidad y la va a introducir dentro de la rama principal de desarrollo, se ejecuten todos los tests generados y se compruebe que todos lo hacen correctamente antes de que dicho código pueda desestabilizar la rama principal de desarrollo.

Además, algunas empresas utilizan un servidor de integración donde implementan procesos de Integración continua que aseguran un control de la calidad del software que se está integrando. Muchas de estas empresas han adoptado *test-driven development* e integración continua sin adoptar todas las prácticas que propone *XP*.

Ventajas

Los programadores no pierden tiempo si algún test falla o se genera algún *bug*, ya que basta con volver a la última versión buena almacenada en el control de versiones.

Los desarrolladores también pueden detectar y arreglar los problemas constantemente, de forma que se evita la introducción de algún *bug* "de última hora" antes de poner el código en producción.

Palabra clave: *Calidad*

Desventajas

Para realizar Integración continua se necesita un tiempo para poner a punto todo el sistema. Además, hay que tener en cuenta que para sacar una ventaja de las pruebas automatizadas se debe tener una gran cantidad de tests desarrollados.

Palabra clave: *Tiempo*

2.3. Los análisis de usabilidad en la web

Los análisis de usabilidad son una parte muy importante del desarrollo de productos software para obtener un mayor éxito y, especialmente, conseguir una mayor productividad de las personas que utilizan el sistema, ya que están realizados de acuerdo a sus necesidades y a su forma de trabajar.

Hoy en día, los análisis de usabilidad son necesarios en cualquier desarrollo en el que existan unas necesidades importantes de interacción con el usuario, comenzando en las etapas más tempranas de la generación de un producto, ya que el avance y el continuo cambio de las tecnologías informáticas hacen que personas que no trabajan en el sector de las Tecnologías de la Información y la Comunicación puedan no adaptarse con tanta facilidad como los que sí lo hacen.

Es más que recomendable la realización de los análisis de usabilidad en proyectos web debido a que las plataformas web otorgan un acceso ubicuo que puede realizarse desde cualquier dispositivo de escritorio o móvil. Esto redunda en que se deben realizar las pruebas correctas con los usuarios finales desde el comienzo para que el grado de aceptación de los mismos sea mayor.

En las próximas secciones se introducen los conceptos historias de usuario, diseño centrado en el usuario, prototipos de la interfaz de usuario y la experiencia del usuario.

2.3.1. Historias de usuario

Las historias de usuario son representaciones de los requisitos software que se escriben en una o dos frases utilizando lenguaje común del usuario. Este lenguaje dependerá del contexto del propio usuario al que van dirigidos los análisis de usabilidad.

En las metodologías tradicionales de desarrollo de software se hacían típicamente casos de uso, que consistían en la representación de las actividades que se deben realizar para llevar a cabo un proceso, pero que no utilizaban un lenguaje cercano al usuario. Las historias de usuario han ido reemplazando total o parcialmente a los casos de uso, haciéndolo más típicamente en aquellos proyectos en los que se realizan análisis de usabilidad.

Las historias de usuario deben ser limitadas, pudiendo ser escritas en pequeños trozos de papel, normalmente adhesivos, para poder utilizarlos posteriormente en los paneles que habitualmente controlan el estado de un proyecto que utilice metodologías ágiles. De este modo, el equipo de desarrollo tiene una forma rápida de administrar los requisitos de los usuarios sin elaborar grandes documentos formales que no hay tiempo de administrar.

Las historias de usuario deben tener las siguientes características:

- Han de ser independientes unas de otras.
- Si no son explícitas, se debe delimitar su alcance mediante pruebas de validación que cumplan sus requisitos concretos.
- Tienen que tener alguna relevancia para los clientes o para los usuarios finales.
- Deben ser cuantificables de tal forma que se pueda estimar el tiempo que llevará desarrollarlas.
- Si son cortas, mejor.
- Si es posible la verificación de sus requisitos funcionales, deben automatizarse para que se validen en cada entrega del proyecto.

Esta técnica se utiliza en muchas de las metodologías mencionadas en el anterior apartado, ya que garantiza el principio de hacer al cliente partícipe al ser él quien debe otorgar al equipo de desarrollo dichas historias de usuario.

Éstas deben tener un orden de prioridades marcadas por el cliente de cara al producto final y, tras la estimación de tiempo, deberían tener una duración de entre diez horas y un par de semanas para poder realizarlas en una iteración.

2.3.2. Diseño centrado en el usuario

El diseño centrado en el usuario es una disciplina que se enmarca dentro del diseño de interfaces de usuario en la rama de la Ingeniería Informática denominada interacción persona-ordenador. Esta disciplina tiene en cuenta las necesidades, los deseos y las limitaciones de los usuarios finales de los productos que se van a desarrollar.

Con el creciente avance de la tecnología y del uso extendido de distintas plataformas con las que el usuario consume dicha tecnología, se ha hecho cada vez más necesaria la formación de expertos en el campo de la usabilidad y la interacción persona-ordenador para que los sistemas software sean utilizados más fácilmente por los usuarios a los que van dirigidos y conlleven una mejor experiencia del usuario y una mayor satisfacción por su parte.

En relación a esta disciplina se ha creado un estándar ISO, denominado *Human-centred design for interactive systems* [ISO10], que describe seis principios básicos para certificar que un diseño está centrado en el usuario. Estos principios básicos son:

1. El diseño está centrado en el conocimiento de los usuarios, sus tareas y su entorno.
2. Los usuarios participan en distintas fases del diseño y del desarrollo.

3. El diseño está dirigido y es refinado por las evaluaciones centradas en el usuario.
4. El proceso debe ser iterativo.
5. El diseño contempla toda la experiencia del usuario con el sistema.
6. El equipo de diseño posee habilidades y perspectivas multidisciplinares.

Dentro de un diseño centrado en el usuario hay que tener en cuenta los siguientes puntos:

- **La audiencia** que son las personas que van a utilizar el producto generado. Por lo tanto, el diseño debe tener en cuenta su edad, cultura, género, etc.
- **El objetivo** que el producto pretende conseguir o los problemas que está intentando solventar.
- **El contexto** y las circunstancias existentes para que se haya creado la necesidad de generar este sistema.

Como en todas las disciplinas, existen herramientas que sirven de apoyo al desarrollador para poder realizar este tipo de diseños. A continuación se explican tres herramientas principales:

- **Personas**

  Una *persona* es un anglicismo utilizado para la representación ficticia de un personaje que posee todas las características del usuario típico al que va dirigido el sistema. Estas *personas* son creadas tras haber realizado un análisis del entorno de la plataforma, de sus usuarios potenciales y de las tareas que realizan éstos en la misma.

  La información de los usuarios se extrae a través de distintas técnicas, como pueden ser la observación del usuario en su entorno de trabajo o la realización de una serie de cuestionarios o entrevistas. Del análisis de los resultados obtenidos se pueden obtener una o varias *personas* ya que puede resultar difícil o imposible introducir todas las características en un único personaje. Hay que tener en cuenta que, como se ha mencionado en el párrafo anterior, las *personas* son usuarios típicos de la plataforma, no una representación del usuario medio.

  Idealmente también se creará un perfil de la *anti-persona* que será una representación del tipo de usuarios a los que la plataforma no va dirigida.

  Una *persona* debe tener un nombre, una ocupación, un rol y unas responsabilidades que irán acompañadas de unas metas, tareas y motivaciones, así como de todo el entorno necesario para comprender su comportamiento frente al sistema que se va a desarrollar y, para dotarle de mayor realismo, debe ir acompañada de una fotografía.

- **Escenario**

  Se debe crear un escenario que represente el día a día con una secuencia de eventos de las *personas* a las que va dirigida la plataforma. Estos escenarios deben descubrir detalles de los usuarios, pudiendo incluir características físicas o emocionales que puedan afectar en el desarrollo.

  Existen distintos tipos de escenarios pero, típicamente, se generan escenarios de tres tipos: el escenario en el que todo le sale bien al usuario, el escenario en el que todo le sale mal, y el escenario medio.

  Los escenarios son más fáciles de seguir para los usuarios ya que involucran a *personas* y una historia que es fácil de seguir.

- **Casos de uso**

  Un caso de uso describe la interacción entre las *personas* y el resto del mundo. En el caso de los diseños centrados en el usuario, los casos de uso se suelen representar como una tabla con dos columnas: a la izquierda el actor principal del caso de uso, y a la derecha, el mundo, que es la representación de todos los aspectos externos al usuario que le proporcionan los elementos que éste necesite para desarrollar satisfactoriamente el caso de uso planteado.

  Típicamente se realiza mediante la formulación de pasos simples que debe realizar el actor y la interacción se basa en un causa-efecto entre el actor y el mundo. Al diseñar los casos de uso, no se deben realizar presuposiciones sobre detalles que no tienen relación con el propio caso de uso.

  Los casos de uso son útiles para el desarrollador ya que se exponen tareas de bajo nivel que se producen en un determinado problema y ayudan a su mejor comprensión y a la obtención de una solución que funcione correctamente.

Existe una extensa bibliografía que menciona el diseño centrado en el usuario. Uno de los libros más influyentes en este tipo de diseños es el libro "*La psicología de los objetos cotidianos*" de Donald A. Norman [Nor90], que junto a Jakob Nielsen es considerado uno de los mayores representantes en el mundo de la usabilidad y la interacción persona-ordenador.

2.3.3. Prototipos de la interfaz de usuario

Los prototipos de la interfaz de usuario son el diseño de las versiones preliminares de un programa que permiten, con una inversión muy pequeña, que el desarrollador explore distintas ideas y conceptos y los evalúe con los usuarios de su sistema.

Tradicionalmente han existido dos tipos de prototipos: los prototipos de baja fidelidad, que suelen ser diseños de la interfaz desarrollados dibujando sobre el papel el aspecto visual de la aplicación; y los de alta fidelidad, que son prototipos completamente funcionales del programa pero que aún no se consideran el producto final.

Los desarrolladores y expertos en usabilidad últimamente utilizan un nuevo tipo de prototipos, denominados prototipos de media fidelidad, que suelen ser prototipos semi-interactivos pero que no tienen todas las funcionalidades completas. Esto se puede lograr mediante programas como Pencil™, Microsoft PowerPoint™ o mediante el uso de tecnologías web de desarrollo rápido utilizando HTML5, CSS y Javascript.

Los prototipos pueden ser generados en cualquiera de las etapas del ciclo de vida del desarrollo, pero idealmente deben ser creados en las primeras etapas y lo antes posible, para que todo el diseño pueda ser evaluado antes de gastar esfuerzo y dinero en desarrollar funcionalidades que puedan no ser tan útiles como se imaginaba.

Jakob Nielsen, doctor en interacción persona-ordenador por la Technical University of Denmark, es una de las personas más influyentes en el campo de la usabilidad. En su artículo titulado "*Paper Prototyping: Getting User Data Before You Code*" [Nie13] explica la importancia de la recolección de datos del usuario lo antes posible en el proyecto, ya que el ahorro estimado puede ser 100 veces más barato si se hace sobre prototipos en papel que si se esperase a la finalización del proyecto para analizarlo.

Antes de continuar explicando los tipos de prototipos y sus particularidades, se debe definir el concepto de interfaz de usuario, que no debe ser confundido ni con el diseño centrado en el usuario (ver sección 2.3.2) ni con la experiencia del usuario (ver sección 2.3.4).

La interfaz de usuario es el método que poseen los usuarios de un sistema software para interactuar con el propio sistema, bien sea mediante un método de introducción de información o manipulación del sistema - donde un ejemplo típico puede ser el teclado o el ratón - o bien mediante la recepción de información por parte del sistema - donde se encuentra por ejemplo la pantalla y el diseño de la propia aplicación.

En otras ramas de la ingeniería existen diversas interfaces de usuario, como pueden ser paneles de botones en el sistema de control de una fábrica, pero en el campo de la Ingeniería Informática existe un grupo distinto de las mismas:

- **Las interfaces gráficas de usuario (GUI)**[6] que generalmente implican que el usuario interacciona mediante los dispositivos de entrada de su ordenador obteniendo una respuesta visual por la pantalla.
- **Los menús** que son un conjunto de acciones que se otorgan al usuario de forma organizada en torno a un elemento común.

[6]del inglés Graphical User Interface, GUI

- **Las interfaces de usuario web (WUI)**[7] que generan su información de salida en formato web, la cual es transmitida a través de internet y puede ser generada utilizando diversas tecnologías. Dentro de éstas podemos encontrar formularios que ayudan a la introducción de información por parte del usuario o cuadros de diálogo que sirven de guía.

- **Los sistemas de reconocimiento de voz** que permiten que el usuario interaccione con el sistema mediante la locución de mandatos de voz que el sistema es capaz de interpretar, realizando la acción solicitada.

- **Las pantallas táctiles** que son utilizadas mayoritariamente en dispositivos móviles y cuyo método de entrada suelen ser los dedos o un lápiz digital.

- **Las interfaces *pen-based*** que permiten la interacción mediante un lápiz que puede poseer un cierto grado de inteligencia y con el que se puede realizar reconocimiento de la escritura del usuario, transformando lo que escribe en órdenes que pueden ser interpretadas.

- **Las interfaces de reconocimiento gestual** que facilitan que el usuario pueda realizar acciones de una forma simple. Un ejemplo de este tipo de interfaces es la conocida *Kinect™* de Microsoft™.

- **Las interfaces de línea de comandos (CLI)**[8] que admiten la entrada mediante la escritura de comandos en un teclado y envían la respuesta en forma de texto que es impreso por la pantalla.

- **Los sistemas hápticos** que generan una sensación de presión real sobre objetos virtuales mediante el uso de sistemas magnéticos o de chorros de aire, dando la sensación de relieve.

- **La realidad virtual** que se aplica mediante el uso de elementos virtuales que simulan un entorno real y con los que el usuario puede interaccionar mediante un avatar que él mismo controla.

- **La realidad aumentada** que genera elementos virtuales basándose en el entorno del usuario, aportando información contextual al entorno real en el que éste se encuentra.

- **Las interfaces compartibles**[9] que permiten la interacción simultánea de varias personas, como por ejemplo una pizarra digital.

- **Las interfaces tangibles** que reaccionan virtualmente mediante la manipulación de objetos reales.

[7] del inglés Web User Interface, WUI
[8] del inglés Command Line Interface, CLI
[9] del inglés Shareable Interfaces

- **Las interfaces vestibles**[10] que permiten llevar puesta la tecnología y responder a las acciones del usuario o de su entorno.

- **Las interfaces robóticas** que posibilitan la sustitución de personas en ciertas circunstancias gracias al grado de libertad e inteligencia que poseen.

- **Las interfaces cerebro-ordenador**[11] que interaccionan con el usuario gracias a los cambios en la actividad de las ondas cerebrales del mismo.

- **Las interfaces afectivas** que intentan descifrar las emociones del usuario y hacer que el sistema se adapte a los sentimientos para obtener respuestas más positivas de éste [Has03].

Dada la variedad de interfaces de usuario y los métodos de interacción, es importante definir prototipos acordes al dispositivo en el que vaya a ser utilizado el sistema que se va a desarrollar. Para ello existen tres tipos distintos de prototipos que pueden ayudar al estudio de dicha interfaz con ayuda de los usuarios:

- **Prototipos de baja fidelidad**

  Estos prototipos son generados en papel e intentan asemejar la apariencia que tendrá la interfaz de usuario. En un prototipo de baja fidelidad no se deben incluir elementos que puedan distraer al usuario (tales como colores) ni formas que puedan asemejarse a algunas funcionalidades particulares del sistema para el que se está construyendo el prototipo (por ejemplo, el uso de los botones predeterminados de un Sistema Operativo o de los globos de notificaciones) [Pro13].

  El prototipo de baja fidelidad, por lo tanto, debe ser lo más "aséptico" posible para poder evaluar correctamente el uso que hace el usuario con los casos de uso y los escenarios planteados.

  Si bien, el prototipo debe tener cierta funcionalidad, por lo que puede ser contemplado con post-its™ u otro tipo de elementos que puedan ser recortados y utilizados para que el usuario tenga una experiencia más cercana a la que finalmente tendría con un sistema completo, y que el entrevistador puede ir manejando y explicando in-situ.

[10] del inglés wearable
[11] del inglés brain-computer interfaces, BCI

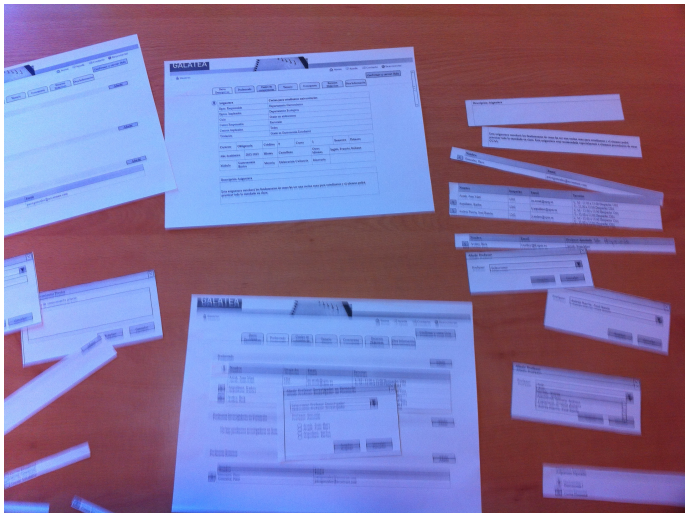

Figura 2.11: Prototipo de baja fidelidad del proyecto Gauss

- **Prototipos de media fidelidad**

  Los prototipos propuestos de media fidelidad incluyen las ventajas de los prototipos de baja y alta fidelidad, pudiendo realizar cambios rápidamente de forma iterativa y pudiendo evaluar determinadas características que pueden resultar difíciles de implementar en prototipos de baja fidelidad en algunos sistemas.

  Gracias al uso de las tecnologías web que permiten desarrollos muy rápidos con poco esfuerzo, utilizando HTML5, CSS y Javascript se puede conseguir un prototipo de media fidelidad en pocas horas y con la funcionalidad justa para evaluar las funcionalidades más complejas conceptualmente. También el uso de herramientas como Pencil, que puede ser instalado de forma independiente o como un *plugin* de Firefox™, que permiten cierta interacción con las interfaces de usuario diseñadas.

- **Prototipos de alta fidelidad**

  Los prototipos de alta fidelidad están desarrollados con tecnologías más cercanas al producto final, con los gráficos ya definidos y prácticamente todas las características implementadas, incluyendo vínculos con fuentes cuasi reales de datos.

  Los prototipos de alta fidelidad son más caros de crear y llevan mucho más tiempo en caso de querer realizar modificaciones, lo que los hace peores candidatos para los análisis de usabilidad tempranos debido al esfuerzo necesario para su desarrollo, pero que si el ciclo de producción está pensado para obtener una versión beta del producto que sirva como prototipo de alta fidelidad, puede ser muy productivo en combinación con cualquiera de los otros prototipos.

Figura 2.12: Pantalla de bienvenida de Gauss

Como se ha podido observar, es recomendable realizar prototipos de las interfaces de usuario puesto que ayuda a obtener mucha información relevante de cara al desarrollo de un producto, y son indispensables si el diseño que se quiere realizar es centrado en el usuario.

2.3.4. Experiencia del usuario

La experiencia del usuario, que no debe confundirse con el término explicado en la sección 2.3.3, se refiere a las emociones que experimenta una persona al utilizar un producto.

La experiencia del usuario se enfoca principalmente en la parte afectiva, los sentimientos de valor y el sentimiento de posesión que inherentemente sentimos los seres humanos ante cualquier producto. Además, también tiene en cuenta si al usuario le resulta práctico o si es fácil de utilizar, así como si éste es más eficiente utilizándolo o no.

Al ser un concepto totalmente subjetivo depende del contexto del usuario y está en constante evolución junto con el estado anímico y los cambios en el entorno del usuario.

Definición.- El estándar ISO 9241-210 [ISO10] define la experiencia de usuario como: “La percepción de una persona y sus respuestas que resultan del uso o del uso anticipado de un producto, sistema o servicio.”

El propio estándar ISO también cita los tres factores que influyen en la experiencia del usuario: el sistema, el usuario y el entorno.

Gracias a este concepto que no sólo evalúa la adecuación de un proyecto a sus usuarios finales, sino también las emociones que éstos sienten al utilizar un determinado

sistema, surge el denominado "Diseño centrado en la experiencia del usuario[12]". Este nuevo concepto de diseño tiene en cuenta todo el sistema, incluyendo la interfaz de usuario, la interacción física y hasta el manual de usuario o el embalaje, ya que forma parte del producto y de la experiencia que tiene el usuario al utilizarlo.

En el concepto de diseño centrado en la experiencia del usuario son muy importantes los siguientes aspectos:

- **El diseño visual**

  La estética o el *look&feel* son importantes para la experiencia del usuario, ya que primordialmente cualquier sistema software interactuará con él de manera visual y siendo visualmente atractivo hará que la experiencia del usuario mejore.

  Es muy importante la combinación de los colores, las imágenes y los símbolos para que el sistema pueda crear una comunicación visual con el usuario.

- **La arquitectura de la información**

  La arquitectura de la información es la ciencia de estructurar y organizar adecuadamente la información en un sistema centrándolo en la usabilidad y permitiendo que el usuario pueda encontrar fácilmente lo que busca.

  La estructuración de la información es vital en una aplicación para que la experiencia del usuario no se vea mermada. Esto se logra estructurando la información, agrupándola y etiquetándola de la manera más coherente posible, para que dicha información no pueda únicamente ser buscada visualmente por el usuario, si no que también pueda ser procesada y buscada mediante un buscador contextual.

- **Usabilidad y accesibilidad**

  No hay que olvidar que hablamos de sistemas cuyo principal objetivo es hacer que el usuario sea eficiente y tenga una mayor satisfacción a la hora de usar el producto. Esto se consigue mediante los análisis de usabilidad que harán mejorar la experiencia del usuario puesto que se sentirá más realizado con la plataforma.

  Todas las personas somos diferentes unos de otros y esto se debe reflejar también en los sistemas software gracias a la accesibilidad, que es la disciplina que estudia las casuísticas de los usuarios con problemas particulares y que adaptan la forma de desarrollar el software para que todo el mundo pueda utilizarlos. También puede servir para que la curva de aprendizaje de cualquier usuario se vea reducida.

En el siguiente capítulo se verá un análisis de la experiencia previa, así como las prácticas introducidas en el desarrollo de la plataforma implementada en este Proyecto Fin de Carrera.

[12]del inglés User eXperience Design, UXD

CAPÍTULO 3

Prácticas propuestas

En el capítulo central de este Proyecto Fin de Carrera se explican las prácticas y las técnicas que el autor propone para utilizar en proyectos similares. Para ello se comienza el capítulo con una sección de análisis, donde se detallan las características del Proyecto, incluyendo sus antecedentes, los requisitos y los métodos de trabajo existentes en su entorno.

Las prácticas propuestas poseen una clara necesidad de mejora de la experiencia del usuario, que se consigue mediante la realización de un correcto análisis de usabilidad y el uso de las técnicas y tecnologías adecuadas por parte del equipo de desarrollo.

A lo largo de las siguientes secciones se podrán evaluar las ventajas y desventajas que se pueden obtener por el uso de las prácticas ágiles, así como la propuesta concreta para la gestión del desarrollo tanto de los análisis de usabilidad como de la implementación de la plataforma y del proyecto en general.

La propuesta de las siguientes prácticas está basada en la experiencia obtenida en áreas más técnicas pero que se han probado y madurado para utilizarlas en un entorno menos técnico centrándolas en los análisis de usabilidad.

3.1. Análisis

En el presente capítulo se presenta un análisis del entorno en el que se ha desarrollado este Proyecto Fin de Carrera, con el fin de que se entienda la necesidad del uso de algunas de las prácticas ya mencionadas en el capítulo 2.

Este proyecto, como ya se ha comentado anteriormente, se ha realizado bajo la dirección del Vicerrectorado de Estructura Organizativa y Calidad por la necesidad de la creación de una herramienta que facilitase el seguimiento de las titulaciones de la UPM para la verificación de sus títulos.

A lo largo de este capítulo se podrá ver la plataforma existente previamente, que ha sido la base utilizada para el desarrollo de esta nueva plataforma, los métodos y tecnologías con las que trabaja habitualmente el equipo de desarrollo de aplicaciones del Rectorado, donde se integran los componentes de la Facultad de Informática, así como las necesidades surgidas de cara a esta nueva herramienta.

3.1.1. Los antecedentes: Europa

A principios del curso 2010-11 el Vicerrectorado de Estructura Organizativa y Calidad pone en marcha una nueva herramienta, denominada Europa, utilizada para la gestión de las guías de aprendizaje, los informes de asignatura, y otros documentos de interés para el seguimiento de la implantación de las titulaciones de la UPM.

Figura 3.1: Pantalla de bienvenida de Europa

Esta herramienta fue construida por una empresa externa a la universidad tomando como modelo las aplicaciones de control de flujo[1], para gestionar no sólo el proceso de aprobación de dichos documentos de seguimiento que involucran a varias personas en distintos roles, sino también para la cumplimentación de las guías de aprendizaje.

Figura 3.2: Imagen ilustrativa del control de flujo en Europa

[1] del inglés workflow

La utilización de un modelo de control de flujo aplicado a la creación de los documentos no era un enfoque correcto según la apreciación del equipo de desarrollo de Gauss, debido a que limita en gran medida la flexibilidad y adaptación del usuario, obligándole a introducir una serie de datos que puede no conocer en ese momento, y no permitiéndole continuar en la cumplimentación de la guía de aprendizaje hasta no haber completado todos los apartados de la sección en la que se encontrase.

Europa utilizaba de manera bastante acertada el concepto de la denominada "Bandeja de trabajo" (ver figura 3.3), en la que el usuario recibía notificaciones sobre los nuevos documentos en los que tenía que trabajar. De esta forma, cuando un usuario se conectaba, podía inmediatamente saber si tenía trabajo por realizar o no dentro de la plataforma.

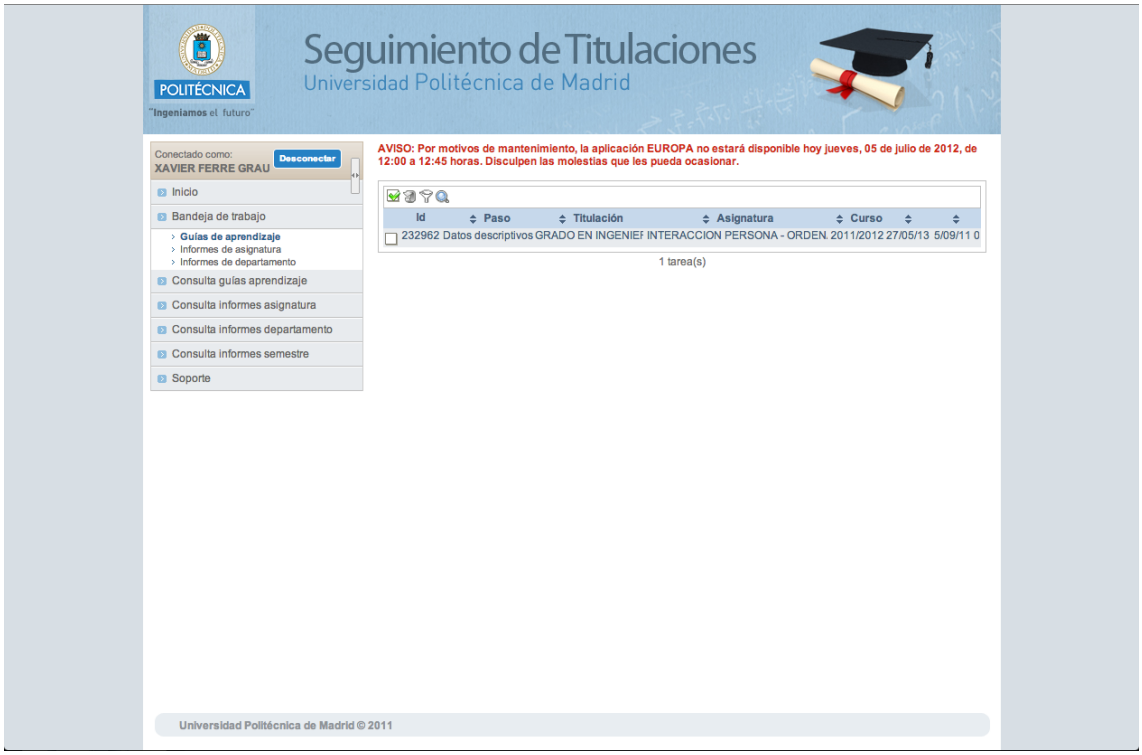

Figura 3.3: Bandeja de trabajo en Europa

Si bien esto fue un acierto, el desarrollo estaba tan centrado en el control del flujo del documento que cuando un usuario finalizaba su trabajo, éste perdía de vista el documento sin posibilidad de saber en qué punto del proceso se encontraba.

Europa, además, era una plataforma desarrollada por una empresa externa a la UPM, por lo que los cambios en los requisitos suponían un coste extra para que la aplicación estuviera conforme a las necesidades del VEOC.

3.1.2. Los requisitos

Durante las primeras fases del desarrollo se realiza un completo análisis de los antecedentes de la herramienta existente y de los nuevos requisitos de los que se quiere dotar a la plataforma Gauss, que sustituirá a la obsoleta Europa.

El equipo de desarrollo formado en la Facultad de Informática fue seleccionado por sus conocimientos en el desarrollo de plataformas web, así como por su experiencia en el campo de la usabilidad y su búsqueda de la mejora constante de la experiencia del usuario.

Desde el principio el enfoque otorgado a la nueva plataforma era radicalmente distinto al propuesto en Europa, comenzando por las bases del diseño y de la formación del equipo de trabajo de la plataforma.

Gauss se convertirá en una aplicación más del entramado de aplicaciones ofrecidas por el Rectorado de la UPM. De esta forma, la nueva aplicación ha sido desarrollada por un equipo multidisciplinar formado por: el equipo de Servicios Informáticos de Rectorado, la Unidad de Calidad y el equipo de desarollo de la Facultad de Informática. Este último grupo, liderado por el autor del Proyecto Fin de Carrera y bajo la supervisión de los autores del proyecto, se ha encargado de desarrollar la aplicación con los requisitos impuestos, en constante comunicación con los Servicios Informáticos que serán los que, en última instancia, se encargarán de su posterior mantenimiento.

Una gran parte del análisis de los requisitos fue la evaluación de las decisiones erróneas que se tomaron en el proyecto Europa, para cambiar el enfoque de la aplicación, así como la determinación de qué puntos fuertes y débiles contenía.

De esta forma se pudo llevar a cabo un completo análisis de usabilidad que se realizaría a los usuarios de forma que se evaluara más fehacientemente si los requisitos eran implementados correctamente, y si las decisiones de diseño tomadas convencían a los usuarios finales de la plataforma, para que la aplicación tuviera una mayor aceptación entre ellos.

Los requisitos de la plataforma incluyen, entre otros, los siguientes:

- Una completa interfaz de administración para realizar el lanzamiento del proceso durante cada uno de los semestres del curso académico, así como una completa gestión de los usuarios y de las guías de aprendizaje.
- La posibilidad de que cualquier usuario pueda delegar a otros usuarios, total o parcialmente, sus responsabilidades dentro de la plataforma.
- La gestión correcta y completa del flujo que siguen las guías de aprendizaje.
- La posibilidad de que un coordinador de asignatura rellene las guías asignadas en el orden que mejor le convenga, sin establecer ningún tipo de bloqueo salvo el propio al finalizar la edición de la guía.

- La posibilidad de agrupar asignaturas, de tal forma que para aquellas asignaturas impartidas bajo la coordinación de un mismo profesor para alumnos de distintos títulos, la guía de aprendizaje pueda ser rellenada una única vez por el coordinador.
- La importación de datos mediante la copia de los mismos de cualquier guía anterior, para que disminuya el proceso de cumplimentación de las guías.
- Un completo sistema de *log* para que cualquier usuario involucrado en el proceso conozca, en todo momento, el estado de las guías que tiene bajo su responsabilidad.

Con estos requisitos de partida, el equipo de desarrollo generó los prototipos de bajo nivel utilizados en los análisis de usabilidad que sirvieron como base para el desarrollo del prototipo de alto nivel de la plataforma.

3.1.3. Las prácticas y tecnologías existentes en la UPM

Desde el primer momento en el que se propone la creación de una nueva herramienta para la gestión del seguimiento de las titulaciones, se tiene en cuenta que el desarrollo debe ser *in-house*, contando para ello con los Ingenieros Informáticos de la UPM, así como con los Servicios de Planificación de Sistemas de Información del Vicerrectorado de Servicios Informáticos y de Comunicación.

Tras haber analizado los requisitos y los antecedentes existentes, así como los plazos propuestos para la creación de esta nueva herramienta, se barajan las distintas posibilidades de desarrollo. Como el grueso del desarrollo se va a realizar desde la Facultad de Informática, pero el equipo de desarrollo es temporal y el mantenimiento quedará a cargo del SPSI, se ha decidido utilizar las tecnologías y herramientas que este Servicio utiliza en el desempeño de sus funciones. De esta forma, la transferencia de conocimiento del equipo de desarrollo al equipo de mantenimiento será prácticamente automática.

Así pues, el desarrollo de la herramienta está realizado utilizando las siguientes tecnologías y herramientas:

- **PHP 5.3** como lenguaje para el desarrollo del *back-end* y del núcleo de la aplicación.
- **Smarty 2** como gestor de plantillas web que permite una alta personalización de las mismas.
- **Oracle 11g** como gestor de base de datos.
- **Toad for Oracle** como *front-end* para gestión de las bases de datos de desarrollo y producción.
- **HTML5** como lenguaje de marcado para las páginas generadas aprovechando algunas de las últimas novedades de su última versión, como el manejo de la subida de ficheros.

- **CSS3** como soporte para el diseño visual de la plataforma.
- **jQuery 1.8** como incorporación definitiva para sustituir las tecnologías *prototype* que se usaban en las herramientas gestionadas por el SPSI.
- **Bootstrap 2** como herramienta para hacer flexible y agilizar el desarrollo de la plataforma, permitiendo adaptar el diseño a distintas resoluciones de forma relativamente sencilla.

La parte del SPSI con la que se ha trabajado y que es la encargada del desarrollo y mantenimiento de nuevas aplicaciones en la UPM, contaba al inicio de este proyecto con tres integrantes que comparten despacho, lo que favorece la interacción entre ellos. Tras la incorporación de los integrantes de la Facultad de Informática hay que sopesar el hecho de que la localización de los mismos dista del resto de miembros del equipo de desarrollo, por lo que la comunicación es un factor crucial a tener en cuenta.

El sistema de control de versiones utilizado por el SPSI es el programa Microsoft Visual Sourcesafe™. Esta herramienta es algo antigua y no permite configuraciones complejas fuera de la red del Rectorado de la Universidad, por lo que ha sido necesario encontrar una solución al problema, otorgando un control de versiones fiable y que permitiera el intercambio de trabajo de manera simple.

Inicialmente, el trabajo desde la Facultad de Informática se ha desarrollado realizando las modificaciones directamente contra el servidor de ficheros FTP que posee el entorno de pruebas. Esta forma de trabajo es bastante anticuada y no permite la flexibilidad requerida ya que los cambios deben ser atómicos y, al no ser capaz de detectar colisiones en los cambios, la comunicación debe ser la prioridad del equipo. Entre las prácticas propuestas se puede encontrar la solución a este problema, que se ha solventado utilizando sistemas de control de versiones distribuidos, en concreto, Git[2].

3.2. Scrum

La primera práctica propuesta para realizar una gestión más eficiente de proyectos con un alto grado de carga en la satisfacción del usuario y con la necesidad de evaluar en profundidad las necesidades del mismo, es Scrum (ver sección 2.2.1).

Como ya se ha explicado en profundidad en qué consiste esta práctica ágil, no se va a entrar en mayor detalle en esta sección, si no que se va a realizar un análisis más centrado en el uso de esta práctica dentro del proyecto y cómo ha servido para detectar mejor los puntos críticos y realizar una planificación más eficiente durante el desarrollo de Gauss.

[2]http://git-scm.com/

Iteración	Fecha inicio	Fecha fin
1	01/10/2012	14/10/2012
2	15/10/2012	28/10/2012
3	29/10/2012	18/11/2012
4	19/11/2012	02/12/2012
5	03/12/2012	16/12/2012
6	17/12/2012	06/01/2013
7	07/01/2013	20/01/2013
8	21/01/2013	03/02/2013
9	04/02/2013	17/02/2013
10	18/02/2013	03/03/2013
11	04/03/2013	17/03/2013
12	18/03/2013	31/03/2013

Tabla 3.1: Tabla inicial propuesta de iteraciones de Gauss

3.2.1. Sprints

Inicialmente la planificación del proyecto estipulaba una duración prevista de seis meses, comprendidos entre Octubre de 2012 y Marzo de 2013. Posteriormente, por problemas externos y una planificación poco realista, se retrasó el final del proyecto varios meses.

En la planificación inicial se calcularon doce iteraciones o *sprints* bisemanales, aunque se fijaron una serie de fechas concretas pasando a ser algunos trisemanales para cuadrarlos con el calendario festivo.

Las fechas propuestas en la planificación se pueden ver en la tabla 3.1.

Al final de las iteraciones se ha tenido una reunión de seguimiento para evaluar las características implementadas y la evolución del proyecto, así como para poder fijar los objetivos que se debían alcanzar en la siguiente iteración.

Diariamente, el equipo de desarrollo hacía una pequeña reunión para verificar los problemas encontrados y las soluciones aportadas y que permitía que el equipo se pusiera a trabajar centrado en la solución de los problemas identificados en la jornada anterior.

Dado que la planificación terminó siendo poco realista, se quiere dejar constancia de la planificación definitiva del desarrollo de Gauss, con el desarrollo prácticamente terminado a falta de la realización de las pruebas definitivas con los usuarios y de su puesta en marcha (ver tabla 3.2).

Iteración	Fecha inicio	Fecha fin
1	01/10/2012	14/10/2012
2	15/10/2012	28/10/2012
3	29/10/2012	18/11/2012
4	19/11/2012	02/12/2012
5	03/12/2012	16/12/2012
6	17/12/2012	06/01/2013
7	07/01/2013	20/01/2013
8	21/01/2013	03/02/2013
9	04/02/2013	17/02/2013
10	18/02/2013	03/03/2013
11	04/03/2013	17/03/2013
12	18/03/2013	31/03/2013
13	01/04/2013	14/04/2013
14	15/04/2013	28/04/2013
15	29/04/2013	12/05/2013
16	13/05/2013	26/05/2013
17	27/05/2013	09/06/2013
18	10/06/2013	23/06/2013
19	24/06/2013	14/07/2013

Tabla 3.2: Tabla final de iteraciones de Gauss

3.2.2. Roles

Los roles se han modificado respecto al planteamiento original de Scrum simplificándolos, dadas las necesidades del proyecto. En este caso se han empleado los roles mostrados a continuación y representados por las siguientes personas:

- **Product Owner** representado por dos grupos diferenciados:
  - El equipo de la Unidad de Calidad del VEOC, que representaba la voz del cliente en el proyecto, pero que no lo hace de manera directa sobre el planteamiento del Scrum, dado que ha sido una práctica adoptada únicamente de forma interna por el equipo de la Facultad de Informática.
  - El autor del PFC, que representaba la voz del cliente dentro de la práctica de Scrum de la manera más objetiva posible, evaluando y priorizando las tareas conforme lo que el cliente real indicaba.
- **ScrumMaster** representado por el autor del PFC como líder del equipo de desarrollo, encargado de solucionar los problemas que impidieran al equipo avanzar con normalidad y con el compromiso de cumplir las “reglas de juego” de Scrum.
- **Equipo de desarrollo** representado por los cuatro miembros del equipo de la Facultad de Informática y que han participado activamente en el desarrollo de la plataforma.

En el uso particular que se ha hecho de los roles de Scrum se pueden identificar algunas fortalezas y debilidades que ha tenido el proyecto:

Fortalezas

Entre las fortalezas, podemos encontrar las siguientes:

- El equipo de desarrollo ha estado más concentrado, pudiendo centrarse en las tareas de desarrollo sin tener que preocuparse por los aspectos menos técnicos del proyecto.
- La comunicación entre el cliente y el líder del equipo de desarrollo ha resultado ser más fluida una vez eliminada la complicación de tener que paralizar a todo el equipo de desarrollo para poder gestionar el proyecto y, más especialmente, tras realizar reuniones más espaciadas para que la apretada agenda del cliente no tuviera que verse modificada.

Debilidades

Entre las debilidades, cabe destacar las siguientes:

- El cliente no ha estado en contacto directo con todo el equipo de desarrollo, lo que provoca que no haya podido ver todos los puntos de vista en todo momento, lo cual podría haber aportado un mayor debate en ciertos puntos y haber acelerado la toma de decisiones, aunque también podría haberlas entorpecido.
- La separación del *Product Owner* en dos grupos de personas va contra la propia filosofía de Scrum pero, en este caso, al tratarse de un proyecto de grandes dimensiones y con agendas del cliente apretadas y con muchos cambios imprevistos, debía realizarse esta separación. Un proyecto con un cliente más comprometido dentro del Scrum puede otorgar resultados más satisfactorios.
- Pese a esta separación, el cliente ha estado informado en todo momento y ha ido viendo la evolución del producto según las iteraciones previstas, que se han ido adaptando a la realidad del proyecto, con lo que sí que ha participado activamente, pero no ha diseñado las historias de usuario ni ha planificado directamente el *Product Backlog*.

3.2.3. Reuniones y herramientas

La gestión de los *sprints* se ha realizado utilizando una pizarra blanca del Laboratorio de Sistemas Operativos de la Facultad de Informática de la UPM, donde el equipo de desarrollo ha desempeñado su trabajo. En la figura 3.4 se puede observar cómo era este tablón de Scrum, el cual estaba dividido en 4 filas y 6 columnas. Al final de esta sección, en la figura 3.5, se puede observar la secuencia temporal en la que las tareas han avanzado durante un *sprint*.

Las filas representan la tipología de las tareas que se han realizado. Esta categorización permite tener una visualización del grupo de tareas donde se debe centrar el esfuerzo del equipo. En este caso las filas representan:

- Tareas de **gestión** que deben realizarse para que el proyecto avance en la dirección correcta.
- Tareas de **documentación** tanto para el afianzamiento de los conocimientos de las tecnologías utilizadas por los miembros del equipo como para la documentación del propio proyecto y la generación de los manuales de usuario, ayudas contextuales, etc.
- Tareas de **desarrollo** que se deben realizar para la generación de la plataforma web. Como es lógico, es la fila que más tareas ha concentrado debido a las dimensiones y particularidades del proyecto.
- Tareas relativas a la **usabilidad** y los análisis que el equipo ha desarrollado. Desde el principio, el equipo ha estado centrado en la realización de diversos análisis de usabilidad para que la plataforma otorgue la mejor experiencia del usuario posible, generando para ello prototipos de bajo nivel y teniendo entrevistas con los usuarios finales de la plataforma.

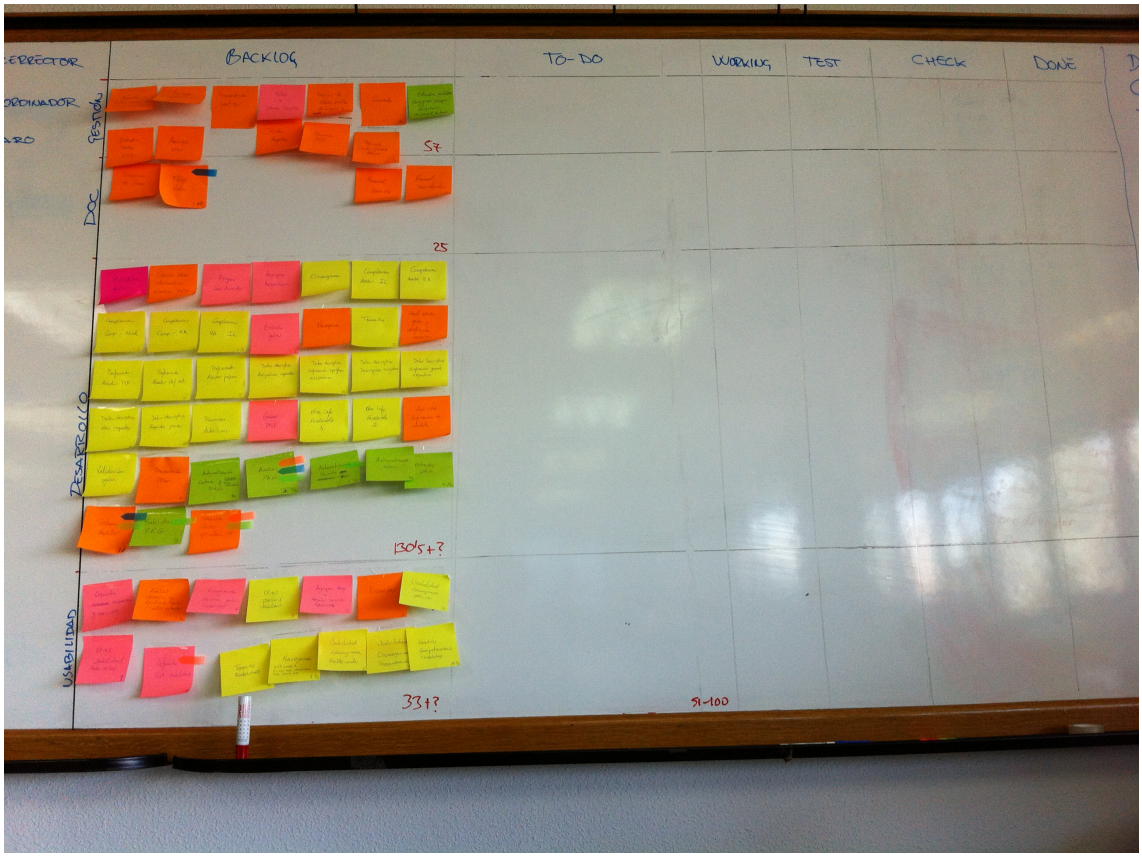

Figura 3.4: Tablón de Scrum del proyecto

Las columnas de la tabla representan el estado de las tareas en una evolución lineal de las mismas dentro del proyecto. De esta forma se distinguen:

- Tareas en ***backlog***, es decir, aquellas que se han identificado y desagregado de las historias de usuario que componen el proyecto y que deben realizarse para alcanzar un objetivo implementado como una característica del sistema.
- Tareas ***to-do*** que son las tareas que se compromete el equipo a realizar dentro del *sprint* en el que se encuentra.
- Las tareas ***working*** en las que se está trabajando activamente en un momento concreto.
- Las tareas en fase de ***test*** que ya han sido codificadas y están en fase de pruebas. Por esta fase pasan únicamente las tareas de desarrollo.
- Las tareas listas para ser comprobadas o en fase de ***check*** para que una persona distinta a la que ha realizado la codificación y las pruebas dé el visto bueno.
- Las tareas finalizadas que van en la columna ***done***.

Al finalizar una iteración se recogen los post-it^{TM} utilizados que estén en la última columna, quedando el resto de tareas pendientes para su realización en el siguiente *sprint*. Se completa la columna *to-do* con un número razonable de tareas para realizar y que, en cierta medida, han sido aprobadas por el cliente.

Típicamente un tablón de Scrum no posee una columna de check, que se ha incorporado a este proyecto con resultados bastante satisfactorios. Dado el distinto grado de experiencia de los componentes del equipo de trabajo y el dimensionamiento del proyecto y sus casuísticas, se entendía necesario el post-procesado de las tareas para comprobar su grado de completitud.

Si las tareas eran realizadas por un único miembro del equipo, cualquier otro miembro del mismo revisaba las características implementadas pudiendo echar marcha atrás la tarea si ésta no pasaba las pruebas pertinentes, suceso típicamente debido a una realización de pruebas poco exhaustivas.

Si las tareas eran realizadas en paralelo por todos los miembros del equipo, éstas eran revisadas por el líder del equipo de proyecto para dar el visto bueno o llevar a cabo las modificaciones o anotaciones oportunas.

Con este sistema de revisión se han solventado muchos problemas, especialmente de codificación, que no han llegado al sistema final gracias a la perspectiva que una persona que no ha realizado el trabajo aporta al mismo.

En las tareas de desarrollo de los prototipos de bajo nivel se han obtenido distintos tipos de prototipos para las pantallas, con una mayor calidad, que han permitido evaluar mejor la opinión de los usuarios finales de la plataforma.

Figura 3.5: Evolución temporal del Scrum del proyecto

Identificación de tareas y responsables

Siguiendo la filosofía de Scrum las tareas eran expuestas en el tablón para que todo el equipo del proyecto tuviera presente el plan de trabajo en todo momento y para que cada miembro pudiera decidir qué tarea realizar tras la finalización de su tarea actual.

Como se ha mencionado anteriormente, se ha utilizado una pizarra blanca con una serie de post-it™ de colores que no identificaban el tipo de tarea, como pudiera pensarse, sino el rol dentro de la plataforma que se vería involucrado en esa tarea concreta.

Si una tarea, como puede ser lógico, involucra a distintos roles, se asignaba el color del rol que mayor importancia tuviera en dicho proceso.

Por lo tanto, una de las primeras tareas que tuvo que realizar el equipo de desarrollo fue la identificación de roles dentro de la plataforma, si bien la diferenciación de los roles en las prácticas ágiles utilizadas se han limitado a los roles más importantes, teniendo un color para el resto de roles. Los roles y colores asignados son los siguientes:

- magenta - Vicerrectorado
- verde - Administrador
- amarillo - Coordinador de asignatura
- naranja - Otros roles

De la misma forma, para que el trabajo estuviera correctamente coordinado entre los miembros del equipo y para saber qué persona estaba trabajando en alguna tarea, se utilizaron post-its™ marcapáginas, de menor tamaño, para identificar a cada una de las personas del equipo de desarrollo.

Esto ha permitido que el desarrollo de las tareas fuera mucho más ágil y que el autor del Proyecto, como líder del equipo de desarrollo, pudiese identificar en todo momento cuál era el estado del proyecto y qué personas estaban trabajando en qué secciones.

3.3. Poker planning

El *poker planning* es una disciplina que permite que un equipo de trabajo estime la carga de trabajo de una tarea concreta estimando su duración en unidades de trabajo (horas, días, etc.).

Esta práctica encaja a la perfección dentro del uso de las prácticas ágiles, y es muy comúnmente utilizada junto a Scrum (de hecho también es conocida como Scrum Poker), ya que cada historia de usuario que se introduce dentro del *Backlog* se cuantifica con un número que es reflejado en la esquina inferior derecha del post-it™ para que todos los miembros del equipo sepan cuánto tiempo les llevaría desarrollar esa tarea.

A la hora de realizar la estimación de una tarea, se pueden utilizar dos tipos de barajas: una baraja especializada que contiene, típicamente, una secuencia de Fibonacci para realizar la estimación, o bien, una baraja normal (tanto española como francesa), donde el peso, salvo las figuras, tienen su valor nominal - ya que las figuras contabilizan como medias unidades o unidades indeterminadas.

Dentro del proceso de valoración de las actividades, existe la figura del moderador, que se encargará de explicar las historias de usuario para que todos los desarrolladores sepan de qué tratan, así como de velar por el cumplimiento de las normas.

Cada desarrollador hace su apuesta con la carta (o cartas) cubiertas, destapándose todas las cartas al mismo tiempo. La persona que realiza la apuesta más alta y la que realiza la más baja pueden expresar brevemente el porqué de su decisión. El proceso se repite hasta lograr un consenso entre las partes.

Figura 3.6: Ejemplo de apuesta en la estimación de una tarea

En el caso de este proyecto se ha utilizado una baraja española donde las cartas del 1 al 9 tenían su valor nominal en horas de esfuerzo, la sota y el caballo toman el valor de medias horas y el rey es un valor indeterminado por no conocerse todo el contexto de la historia de usuario, o bien porque la estimación era demasiado grande. Durante una apuesta se podía acumular cualquier número de cartas para realizar diversas estimaciones, tal y como se observa en la figura 3.6, hasta un máximo de 46 horas de esfuerzo si se juntaban todas las cartas.

Los miembros del equipo de desarrollo realizaban una primera estimación de cada una de las tareas a realizar, explicándose los motivos de la más alta y la más baja, pero descartando siempre ésta última por querer realizar una evaluación pesimista, para siempre posicionarse en el "peor caso realista" para una tarea. En la segunda ronda, las valoraciones debían estar entre el resto de votaciones, quedándose con la más alta de esta ronda, salvo que existiese un consenso verbal para adoptar cualquier otra duración como válida.

Como ya se ha comentado, esta estimación era plasmada en el post-it^{TM} correspondiente. Cuando la persona encargada de la realización de una tarea la finalizaba, procedía a restar el número de horas que había empleado en desarrollarla, quedando reflejado el tiempo real de esfuerzo en el desarrollo de una tarea. Si al finalizar su jornada laboral no terminaba la tarea, debía restar el número de horas dedicadas para mantener actualizada la estimación de la tarea (ver figura 3.7).

Dado que el método consiste en restar al valor estimado el valor de horas reales, si una tarea excede de las horas previstas para su realización, ésta quedaría finalmente con una cantidad negativa de horas, permitiendo saber cuánta desviación ha habido finalmente en su desarrollo y cuál ha sido la duración real del mismo.

Figura 3.7: Una tarea finalizada del proyecto con una estimación de 4 horas y media y 3 horas de desarrollo realizadas

Al finalizar un *sprint* la información de todas las tareas finalizadas se utilizaba para realizar una mejor aproximación en el esfuerzo de las tareas del siguiente *sprint*.

Esta técnica ha permitido que el equipo tuviera muy claro cuáles eran las tareas más importantes en el desarrollo de todo el ciclo de vida del producto, siendo capaces de valorar correctamente las tareas más pesadas con una estimación mayor. Gracias a esto, todo el equipo estaba alineado y conforme con las estimaciones y consciente de la estimación pesimista, por lo que el trabajo no debería, en la mayoría de las ocasiones, superar esa duración.

3.4. Pair programming

La práctica del *pair programming* explicada anteriormente en la sección 2.2.4 ha sido de gran utilidad durante el desarrollo del proyecto, tanto en la elaboración de los prototipos y de los test de usabilidad, como en la parte más técnica del proyecto.

Durante el desarrollo de todo el proyecto se ha utilizado esta técnica en el denominado modo promiscuo, dentro de las limitaciones horarias del trabajo de cada persona que no tenían por qué coincidir, de tal forma que una persona intentara siempre trabajar con otra distinta que la vez anterior, para favorecer el enriquecimiento y el intercambio de ideas entre todos los miembros del equipo, adoptando distintos roles a la hora de desempeñar su labor.

A la hora de realizar los prototipos de bajo nivel la supervisión de otra persona, en momentos puntuales, ha ayudado a detectar fallos en la redacción de frases, el diseño de una interfaz concreta o ayudado a aumentar la productividad aconsejando una determinada forma de trabajar.

El desarrollo de los tests de usabilidad y de los escenarios que posteriormente se iban a evaluar con los usuarios finales de la plataforma, ha permitido una mayor variedad de preguntas y una mejor forma de valorar los distintos aspectos que se querían evaluar de los usuarios, para obtener resultados más relevantes.

Aunque el uso de esta técnica ha sido muy positivo en cuanto a los análisis de usabilidad se refiere, es mucho más efectiva a la hora de realizar un desarrollo técnico en una plataforma web. Dada la idiosincrasia de los integrantes del equipo de desarrollo, en el que la experiencia con las distintas tecnologías a utilizar era bastante dispar, el ritmo de aprendizaje y la adaptación al ritmo de trabajo del resto de los integrantes han sido mucho más rápidos respecto a las necesidades del proyecto, gracias a la observación y los comentarios del “observador”.

3.5. Integración continua

Los proyectos software pueden tener diversos problemas a lo largo de su desarrollo pero, sin lugar a dudas, uno de los más importantes es un diseño mal planteado que provoque que "las piezas del puzzle no casen entre sí". Esto quiere decir que, si bien una programación modular y bien estructurada facilita mucho el desarrollo del software, puede ser una pesadilla si desde el principio no se ha pensado en las consecuencias de un diseño que no plantease la integración de todas las partes al final.

Es por este tipo de problemas que surge la necesidad, en los proyectos reales, de utilizar técnicas de integración continua que realicen comprobaciones sobre el estado de la plataforma mientras se está desarrollando para comprobar que es consistente. Si el cliente en un momento determinado decide realizar un paso a producción de su producto, éste puede ser lanzado sin tener esas complicaciones asociadas.

En este proyecto la integración continua ha sido implementada gracias al sistema de control de versiones utilizado, que automáticamente realizaba la integración de los componentes poniéndolos en el sistema en marcha para poder realizar las comprobaciones de las funcionalidades implementadas y ver cómo afectaban al resto de componentes asociados.

Aunque se trate de un proyecto web, se puede realizar integración continua completa realizando tests automatizados utilizando la técnica de TDD y herramientas de tests unitarios como PHP Unit o Selenium.

Debido al tiempo limitado para realizar el proyecto, no se ha realizado una Integración continua exhaustiva con tests unitarios, ya que por el acoplamiento que había que realizar con los sistemas que ya funcionaban dentro del entramado de aplicaciones de la UPM, no ha sido posible, pero todos los cambios eran versionados utilizando Git entre todos los miembros del equipo y subidos automáticamente por FTP al servidor en el que se realizaban, eso sí, de forma manual, las pruebas correspondientes a las características implementadas.

3.6. Control de versiones distribuido

Inicialmente, tal y como se ha explicado en la sección 3.1.3, el desarrollo del proyecto se realizaba a través de la modificación de los ficheros fuente directamente en el servidor FTP, lo que conllevaba problemas de sincronización de los ficheros y pérdida de modificaciones realizadas. Esta situación podría darse, por ejemplo, si una persona realizaba una modifiación y otra, al mismo tiempo, realizaba su propia modificación, o si alguna de las personas no cerraba el fichero tras utilizarlo pensando, erróneamente, que su versión era la última válida en el sistema.

Este problema provocó una serie de, por suerte, pequeñas pérdidas de información y una inseguridad creciente a medida que el proyecto avanzaba por la posibilidad de perder una mayor cantidad de información, máxime cuando el resto del equipo técnico realizaba modificaciones en remoto y el equipo de desarrollo no tenía constancia de dichas modificaciones, hasta que se encontraba un error incoherente con los cambios realizados.

Por ello se tomó la decisión de utilizar Git como sistema de control de versiones distribuido, bien conocido por el autor de este Proyecto Fin de Carrera, implementado sobre la propia máquina en la que ha desarrollado su trabajo, incluyendo las modificaciones necesarias para que los ficheros fueran copiados al servidor de desarrollo mediante FTP.

Gracias al uso de un sistema de control de versiones distribuido, cada uno de los miembros del equipo de desarrollo poseía una copia válida del entorno de trabajo, salvo por sus propias modificaciones locales, que tendían a ser mínimas debido a que las pruebas de las funcionalidades empleadas debían ser realizadas desde el servidor de desarrollo y para esto había que realizar la subida de los ficheros a través de Git.

El efecto en el desarrollo ha sido muy positivo, gracias a que se ha dinamizado el desarrollo y se ha añadido un componente de tranquilidad para no perder en ningún momento datos que pudieran llevar al fracaso del proyecto.

3.7. Datos en los análisis de usabilidad

Cuando se realizaron los prototipos, se pensó en una serie de cuatro escenarios para poder probar distintas funcionalidades del sistema con diversos usuarios y sin abusar de su tiempo y su paciencia, de forma que los tests fueran óptimos para el equipo de desarrollo y que las conclusiones obtenidas fueran válidas.

También, debido al gran número de usuarios esperados en cada uno de los centros, un sistema con diversos prototipos y varios escenarios daba una mayor flexibilidad para detectar más rápidamente errores y poder tomar mejores decisiones en cuanto al desarrollo de algunas de las funcionalidades de la plataforma, que resultaban ser menos críticas por estar más estandarizadas, no sólo en Internet, sino también en las aplicaciones de la UPM.

El mayor problema de este tipo de análisis de usabilidad es la gran cantidad de datos que se recogen durante su desarrollo, más aún cuando el número de usuarios, como en este caso, era elevado, de forma que el manejo de información era bastante tedioso.

Por ello, se pensó en una forma rápida y sencilla de almacenar y categorizar toda la información recibida por parte de los usuarios durante las entrevistas realizadas, para poder explotarla a posteriori obteniendo datos concretos sobre el análisis que ayudaran a enfocar mejor el desarrollo de la interfaz de usuario y que pudieran, a su vez, servir en futuros tests de usabilidad con los mismos usuarios.

De esta forma se creó una base de datos que almacenara en tres tablas diferentes datos sobre los análisis de usabilidad:

- **Comentarios**

  Se almacenaban los comentarios realizados por los usuarios, identificando el usuario, el prototipo, el escenario, así como la fecha y el comentario realizado, que eran introducidos en la base de datos de forma normalizada para poder realizar un mejor tratamiento de los datos.

- **Sugerencias**

  Se almacenaban las sugerencias de los usuarios para poder tenerlas en cuenta de cara a las posibles mejoras presentes y futuras de la plataforma. También se guardaba el registro de la persona, la fecha y la sugerencia realizada.

- **Usuarios**

  En esta tabla se almacenan los usuarios con su nombre y el centro al que pertenecen, así como su dirección de correo electrónico de contacto.

Gracias al almacenamiento de los datos éstos se han podido explotar de una manera más eficiente, generando estadísticas de los comentarios más repetidos por prototipo y entre todos los prototipos, que fueran indicativos de un posible problema en el diseño que condujera a una modificación del mismo, o bien que mostraran que el prototipo utilizado era idóneo para una funcionalidad concreta.

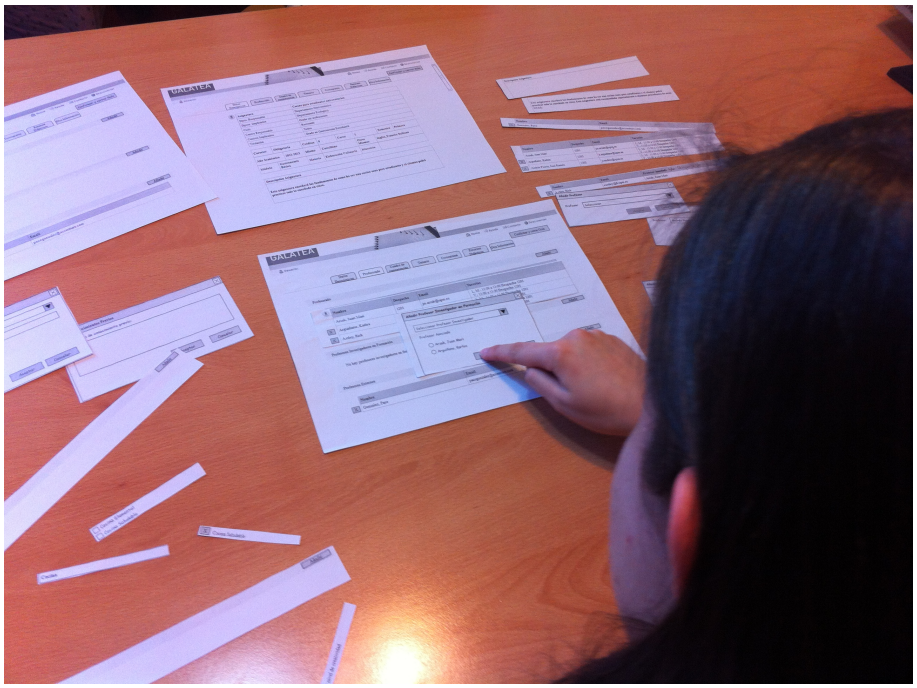

Figura 3.8: Entrevista utilizando un prototipo de baja fidelidad

Así mismo, tras la recopilación de los datos durante los *tests* de usabilidad de los prototipos de bajo nivel y el desarrollo del prototipo de alto nivel, éstos han podido ser explotados de forma que, en las visitas a los usuarios para la realización de análisis de usabilidad con el prototipo de alto nivel, se consiguiera un mayor acercamiento y se creara un compromiso con el usuario.

Gracias a la posibilidad de evaluarle con el mismo escenario, se refuerza el análisis de la capacidad del usuario para recordar la plataforma y se abre la posibilidad de poder analizar con el usuario los errores más repetidos durante el primer análisis de usabilidad y, de esta manera, poder debatir sobre las sugerencias propuestas o los cambios realizados por temas técnicos sobre el prototipo de alto nivel.

El almacenamiento de estos datos y su posterior tratamiento han resultado ser, por tanto, un gran acierto en el desarrollo de los análisis de usabilidad y ha permitido una mejor planificación de las tareas del Scrum al identificar más concretamente los errores cometidos y los puntos críticos del sistema.

CAPÍTULO 4
Conclusiones

En este capítulo se muestran los problemas que se han encontrado con la adaptación de las prácticas ágiles propuestas en el capítulo 3. También se muestran las conclusiones de las ventajas y desventajas que ha supuesto su uso para el desarrollo del proyecto en general y para la realización de los análisis de usabilidad en particular.

4.1. Problemas encontrados

Durante el desarrollo del proyecto, como suele ser habitual, se han encontrado múltiples problemas de índole técnica, que no serán abordados en este Proyecto Fin de Carrera, para poder analizar con una mayor profundidad los problemas encontrados debidos al uso de las prácticas ágiles propuestas.

Se ha optado por dividir el presente capítulo en las mismas secciones existentes en el capítulo 3, de forma que sea más fácil identificar los problemas encontrados en cada una de las prácticas concretas utilizadas.

4.1.1. Scrum

Durante la utilización de Scrum se han detectado varios problemas por lo que su aplicación no ha sido del todo completa.

Como ya se comentó en el apartado 3.2.2, los roles planteados en esta práctica no se han seguido tal y como se muestran en su versión original. Como el *Product Owner* se ha representado en dos grupos de personas, ha conllevado una serie de modificaciones en su tratamiento que, de forma efectiva, ha afectado al enfoque inicial en el uso de esta práctica ágil.

Pese a que en el planteamiento inicial el autor del Proyecto realizaba su labor en la práctica de forma objetiva evaluando y priorizando las tareas conforme el cliente indicaba, éste se ha visto influido por más factores a parte del propio cliente a la hora de evaluar dichas tareas, a saber:

1. Los conocimientos técnicos encontrados en el desarrollo, que han hecho que el orden de las prioridades haya variado, situando en primer lugar las que mayor riesgo y esfuerzo de desarrollo suponían.

2. Los problemas técnicos encontrados en alguna tarea, que han hecho que dichas tareas perdieran prioridad respecto a otras menos importantes.

3. El entorno del equipo de desarrollo, que ha fomentado que unas tareas puedan ser más prioritarias sin serlo para el cliente, basándose en el estado de ánimo de los miembros del equipo, si para éstos supone realizar antes tareas más cómodas de llevar a cabo.

De esta forma, no se ha conseguido una implementación fidedigna de las tareas en el orden deseado por el cliente, sino que éste ha sido modificado ligeramente dentro de unos parámetros normales que no han influido negativamente en el desarrollo del proyecto.

Por otra parte, no todo el equipo conocía inicialmente esta práctica, lo que ha supuesto la dedicación de un pequeño tiempo de documentación y adaptación a la nueva forma de trabajo para ejecutarla exprimiendo toda su funcionalidad.

También hay que tener en cuenta que los horarios variables de los miembros del equipo de desarrollo, por sus obligaciones académicas, no han hecho posible el normal desarrollo de las reuniones diarias para realizar la planificación, y que las reuniones fuera del puesto habitual de trabajo, cuando han coincidido con un cambio de *sprint*, han provocado cambios en la estructura de las reuniones realizadas.

Pese a todo esto y sin lugar a dudas, el mayor problema y a la vez la mayor virtud de esta práctica son los imprevistos. Éstos conllevan modificaciones sobre la planificación, pero gracias a su sistema iterativo de corto plazo, los cambios necesarios y la adaptación a los mismos, hacen que esta práctica sea más fluida que otro tipo de ciclos de vida y prácticas existentes.

4.1.2. Poker planning

La técnica del poker planning ha sido muy útil a la hora de cuantificar el esfuerzo destinado a las distintas tareas planteadas.

El mayor problema encontrado en el poker planning ha sido la inexperiencia del equipo a la hora de tomar decisiones de esfuerzo y dedicación para muchas de las tareas, bien por encontrarse con problemas complejos y que son más difíciles de cuantificar, o bien por la gran cantidad de inconvenientes de los nuevos campos iniciados en esta plataforma, que ha supuesto un reto en la planificación de los mismos.

Otro problema destacado es que, dada la similaridad de los perfiles de los miembros del equipo de trabajo, la estimación de las tareas ha sido muy poco dispar en términos generales, lo que ha otorgado poca confianza en la estimación de una tarea debido a la corta experiencia en gestión por parte de los componentes del equipo.

4.1.3. Pair programming

No se ha encontrado ningún tipo de problema asociado al uso de esta práctica, sino más bien todo lo contrario. Como se explicará en la sección 4.2.3, el uso de *pair programming* ha sido muy positivo entre todos los integrantes del equipo de desarrollo.

4.1.4. Integración continua

La integración continua no se ha podido incorporar completamente, tal y como se ha visto en el apartado 3.5, ya que no se han realizado tests unitarios automatizados que pudieran ser incorporados al sistema de control de versiones utilizado.

Las ventajas de la realización de la integración continua son evidentes gracias a la experiencia en otros proyectos similares, si bien las limitaciones existentes dentro del desarrollo de este proyecto han supuesto que no se pudiera hacer una integración continua al uso.

Durante el despliegue de la plataforma se han encontrado fallos debidos a la sincronización de los ficheros, especialmente antes del uso de un control de versiones adecuado, lo que ha provocado que la tarea de desarrollo se convirtiera en algo arduo y tedioso en muchas ocasiones para poder encontrar los fallos generados sin motivo aparente.

En este caso, al realizar una integración contra un servidor de pruebas realizando la transferencia de los ficheros mediante el protocolo FTP y por la localización geográfica dispersa de los equipos de trabajo, se han observado conflictos y colisiones entre los ficheros incorporados a dicho servidor.

También cabe destacar que las diferentes configuraciones de los servidores de pruebas y producción han provocado muchos errores en la codificación de las funcionalidades. Estos errores hubieran sido difíciles de detectar aún con el uso de algunas de las técnicas de integración continua, aunque podían haber sido más fácilmente detectables con tests unitarios y una configuración homogénea en los distintos servidores utilizados.

4.1.5. Control de versiones distribuido

El uso del control de versiones distribuido ha sido un gran acierto para la consecución de los objetivos.

Aún así, su uso ha provocado algún que otro problema, ya que la implantación inicial se realizó sobre una máquina Windows, lo que provocó varios problemas de sincronización entre los componentes y una ardua tarea de documentación para el correcto funcionamiento del mismo.

Además, durante la sincronización de los ficheros, cada miembro del equipo descargaba y volvía a actualizar los cambios ya realizados, con la consiguiente inefectividad en cuanto a ancho de banda y recursos utilizados.

La migración a una máquina en un entorno Linux del repositorio mitigó todos los fallos y la sincronización ha resultado ser mucho más fiable.

4.1.6. Datos en los análisis de usabilidad

La recopilación de los datos obtenidos durante los análisis de usabilidad ha sido otro gran acierto para la consecución de los resultados obtenidos, si bien ha venido acompañada de un par de problemas que podrían subsanarse en futuros proyectos:

- La introducción de los datos se realizaba tras la finalización de las entrevistas y los análisis de usabilidad con los usuarios, incluso con varios días de diferencia, por lo que los datos almacenados podían no coincidir fidedignamente con los recogidos. Así mismo, llevaban un alto grado de subjetividad ya que cada miembro del equipo podía interpretar los datos de una forma que luego era mezclada y recopilada para evitar las posibles duplicidades.
- La gestión de los datos se realizaba sobre una base de datos a través de una plataforma web, por lo que los datos no pasaban ningún tipo de validación automática, lo que ha podido provocar errores no voluntarios por parte de los miembros del equipo. Además, la visualización de los datos no era la más cómoda posible, por lo que el filtrado ha resultado ser bastante manual y rústico.

4.2. Puntos fuertes

A parte de los problemas encontrados en el uso de estas técnicas, que han sido en su mayoría determinados por factores que pueden ser inherentes a este proyecto, se han observado una serie de puntos fuertes que pueden ser aplicables a otros proyectos similares.

4.2.1. Scrum

El punto más fuerte que posee Scrum es la fidelización inmediata que se consigue entre el cliente y su producto, ya que es el principal recurso del mismo y, por ello, se obtienen todas las ventajas que éste puede aportar al proyecto. Esto redunda en una mayor satisfaccción y mejores resultados.

Desde el punto de vista de la gestión del proyecto, el uso de Scrum facilita la visibilidad del estado del proyecto y el análisis minucioso de las tareas que faltan por desarrollar en un *sprint* concreto.

El uso del *burn down* es especialmente útil para poder comprobar la viabilidad de un proyecto dentro de los plazos establecidos, lo que permite observar visualmente la evolución que ha sufrido el proyecto y su proyección estimada hasta la fecha establecida.

Además, las propuestas de cambio realizadas sobre el marcado de las tareas con post-its^{TM} de diferentes colores, expuestas en la sección 3.2.3, facilitan las labores de revisión por parte del líder de proyecto. En este caso, es una modificación fácilmente escalable y modificable para cubrir distintos tipos de proyectos, de tal forma que el coordinador de un equipo de trabajo pueda, en cualquier momento y de forma visual, saber qué impacto tendrán en la plataforma las tareas en las que se está trabajando en un momento dado pero, sobre todo, qué personas están trabajando en dichas tareas y la duración estimada total y la que le resta a cada historia de usuario.

Por otra parte, no hay que olvidarse de los *sprints*, que son el factor principal de la práctica Scrum, ya que, frente a otros tipos de ciclos de vida en el software, permiten una flexibilidad de desarrollo que posibilita la realización de cortas iteraciones, logrando una planificación mucho mejor del proyecto e, indudablemente, adaptándose mejor al cambio y a los imprevistos.

Todo esto hace que Scrum sea una práctica muy a tener en cuenta por parte de los líderes de proyectos, ya que permite que se puedan generar informes más eficaces sobre la planificación y genera una mayor satisfacción del cliente.

4.2.2. Poker planning

Entre los puntos fuertes destacados que se han observado del uso de la técnica del *poker planning*, se debe destacar la adaptabilidad a nuevas tareas incorporadas en posteriores iteraciones, ya que el equipo ha podido realizar una mejor estimación de los tiempos basándose en la propia experiencia del proyecto.

Las estimaciones realizadas teniendo en cuenta la propia experiencia del proyecto son, en teoría, mucho mejores respecto a las basadas en las experiencias de cada miembro del equipo, ya que se tienen en cuenta otros factores como el trabajo en equipo entre los miembros y las experiencias propias relacionadas con el entorno de desarrollo.

Lamentablemente no se ha podido cuantificar en este proyecto la mejora de las estimaciones a lo largo del mismo debido a que no se han realizado mediciones exhaustivas de tiempos para cada tarea.

4.2.3. Pair programming

Como se ha comentado en la sección 4.1.3, el uso de la práctica del *pair programming* ha sido muy beneficiosa para la evolución del proyecto.

En primer lugar, la mayor experiencia en el desarrollo de proyectos basados en web del autor de este Proyecto, ha servido de ayuda al comienzo del mismo para la mejor integración y adaptación a la forma de trabajo de cada uno de los miembros del equipo de desarrollo, ya que ha apoyado a éstos como guía en la generación de nuevas funcionalidades.

Por otra parte, a lo largo de la ejecución del proyecto y comenzando por el desarrollo de los prototipos de bajo nivel, se ha mejorado la sintonía entre los miembros del equipo de desarrollo gracias al uso de esta técnica. Además, en lo referente al desarrollo de los prototipos, se han intercambiado diversas ideas para la generación de tres tipos bien diferenciados que han permitido un mejor análisis de las necesidades de los usuarios.

También hay que tener en cuenta que, tras la incorporación de un nuevo miembro al grupo de trabajo menos experimentado que el resto, la curva de aprendizaje de éste se ha visto reducida drásticamente, disminuyendo su tiempo de integración en el equipo, lo que ha permitido que resultara ser muy útil para agilizar el desarrollo de ciertas funcionalidades.

Por último, hay que tener en cuenta que esta técnica es fundamentalmente práctica en el entorno de la codificación de las distintas características de la plataforma, ya que da lugar a código mejor estructurado, más legible y, lo que es más importante, con una menor cantidad de errores y especialmente con un tiempo de detección y depuración de fallos mucho menor, lo que acelera el tiempo de desarrollo y asegura la mejor calidad del producto final.

4.2.4. Integración continua

Como ya se ha comentado anteriormente, las técnicas de integración continua no han sido implementadas en su totalidad y se basan en un subconjunto de las prácticas muy poco automatizadas pero que han servido para generar un protocolo de actuación en la prueba de la implementación de las tareas.

Gracias a la implementación parcial de esta práctica, se ha observado su potencial y se estima que su principal ventaja es la contribución a la disminución de la cantidad de errores del producto final, así como a la reducción considerable de su tiempo de pruebas.

4.2.5. Control de versiones distribuido

El uso de un sistema de control de versiones distribuido conlleva muchas ventajas dentro de un entorno de desarrollo multidisciplinar y con varios equipos que cooperan por conseguir un objetivo común.

Los sistemas de control de versiones sirven, especialmente, para llevar un completo control sobre las distintas versiones de todos los documentos de un proyecto, no únicamente ficheros fuente y scripts de codificación. Esto permite llevar un completo histórico de todas las modificaciones realizadas a cada documento y, en caso de cometer algún error o querer volver a una versión anterior, facilitar dicha marcha atrás.

En este caso, es más importante si cabe el uso de un sistema distribuido como Git (usado en este proyecto) o Mercurial, que permita que cada persona posea una copia completa del entorno de desarrollo. Así, si alguno de los equipos informáticos sufriera algún percance, se perderían únicamente las modificaciones locales realizadas por su propietario.

Además de llevar un completo control de las versiones de todos los documentos del proyecto, ha permitido realizar una integración continua copiando automáticamente las modificaciones a los servidores de pruebas mediante el uso del protocolo FTP, pudiendo probar así en el entorno real los cambios realizados.

4.2.6. Datos en los análisis de usabilidad

El almacenamiento de forma estructurada en una base de datos de los usuarios involucrados en los análisis de usabilidad realizados, así como de todos los comentarios y sugerencias recibidos durante los análisis de usabilidad y las entrevistas realizadas, ha sido de gran utilidad para su posterior tratamiento.

El almacenamiento y tratamiento de estos datos ha servido para identificar varios puntos clave para el desarrollo de Gauss:

- La mejor solución propuesta entre los tres prototipos presentados a los usuarios para cada una de las pantallas de las que se compone la plataforma. Así, se tenía la certeza de haber elegido la opción mayormente aceptada.
- Los fallos más comunes entre los usuarios finales de la plataforma que no eran capaces de adecuar su trabajo al diseño planteado por el equipo de diseño. De esta forma, se han podido identificar más fácilmente y de una manera cuantificable el número de errores cometidos sobre una misma acción y se han determinado los pasos que produjeran una mejora sustancial para disminuir su número.
- Las sugerencias más similares entre sí para poder agruparlas en conjuntos de mejoras que llevar a cabo durante el desarrollo de los prototipos de alto nivel y del producto, así como líneas futuras de desarrollo mucho más complejas de desarrollar.

Así, el diseño final de la plataforma ha sido el más adecuado a los requisitos solicitados sin reñirse con la funcionalidad y el aspecto visual demandados por los usuarios finales.

4.3. Conclusiones generales

El uso de las prácticas ágiles ha supuesto un gran avance en el desarrollo de este proyecto, debido principalmente a su adaptabilidad a los cambios, teniendo en cuenta que los requisitos han sido muy cambiantes a lo largo de todo el desarrollo.

La definición tardía de los requisitos definitivos, así como el constante cambio debido a la diversidad de planes de estudios y centros involucrados en el desarrollo de esta plataforma, han sido claves para el éxito de una gestión con prácticas ágiles frente a una gestión con metodologías menos innovadoras o la adaptación del proyecto a los ciclos de vida del software tradicionales.

Esta versatilidad ha permitido que, pese a los tiempos de desarrollo tan limitados que se tenían estipulados, la desviación haya sido mucho menor de lo que podía haber sido mediante el uso de otro tipo de técnicas.

Por otra parte, la posibilidad para los líderes del equipo de desarrollo de poder visualizar completamente el estado del proyecto y poder cuantificar cuánto tiempo puede restar al desarrollo en una situación ideal en la que no cambiaran los requisitos, ha sido crucial para la toma de decisiones estratégicas de cara a la mejora continua del producto sin sacrificar enormemente los tiempos de desarrollo.

Cabe destacar el poco tiempo de adaptación del equipo de desarrollo al hábito en el uso de estas prácticas por ser muy fácilmente adaptables a prácticamente cualquier entorno de trabajo y proyecto, y por permitir la consecución de resultados muy rápidamente, lo cual genera un plus de motivación para el equipo y para el cliente.

CAPÍTULO 5
Líneas futuras

El presente Proyecto Fin de Carrera pretende sentar las bases de futuras líneas de trabajo que permitan tanto la creación de nuevos Proyectos Fin de Carrera, como abrir posibles futuras líneas de investigación en el campo de las prácticas ágiles adaptadas a proyectos con una gran carga de análisis de usabilidad por ser complejos o tener unas necesidades muy particulares en cuanto a su diseño.

5.1. Toma de datos

Las primeras líneas de desarrollo que se pueden realizar van en conjunción a una toma más exhaustiva de los datos referentes al uso de estas prácticas ágiles. De esta forma, se puede realizar un análisis comparativo completo con datos reales de los tiempos empleados en el desarrollo de proyectos que usan este tipo de técnicas frente a otros proyectos con metodologías de desarrollo más pesadas.

Entre los tiempos que se pueden tomar podrían estar los siguientes:

- El tiempo total en un proyecto de similares características.
- El tiempo total en el desarrollo de una tarea estándar que pueda ser identificable en los distintos proyectos.
- El tiempo total dedicado a reuniones de coordinación a lo largo de todo el proyecto.
- El número de errores encontrados en el código en cada fase de desarrollo.
- El tiempo medio de resolución de los errores sin afectar al ciclo de vida del software adoptado en cada proyecto.

Con la realización de esta toma de tiempos, se puede presentar un estudio objetivo sobre la mejora real del uso de este tipo de prácticas y concluir si merece o no la pena su implantación en este tipo de proyectos.

5.2. Estándar de marcado de tareas

Otra línea de desarrollo futura puede ser la estandarización de un sistema de marcado de tareas (o historias de usuario) que permita poder obtener entre diversos proyectos una línea común de marcado de las distintas características a desarrollar, que ofrezcan la posibilidad de equiparar el avance de dos proyectos totalmente distintos.

Esta estandarización podría conllevar muchas ventajas a las organizaciones que desarrollan software, gracias a que un equipo de trabajo puede iniciar un nuevo proyecto con una estructuración lógica y razonada de las tareas y que éstas sean muy fácilmente identificables.

Además, permite que una persona cambie de proyecto en momentos puntuales por decisiones estratégicas y que su adaptación al nuevo entorno de trabajo sea menos traumático, permitiéndole olvidarse de aprender y conocer el sistema de gestión del nuevo proyecto, centrándose única y exclusivamente en el desarrollo del producto.

5.3. Implementación de una plataforma de gestión

A pesar de la existencia de herramientas de control separadas para la mayoría de las prácticas citadas, sería recomendable el desarrollo de una herramienta software que permita la gestión automatizada e independiente de las tareas a realizar siguiendo las prácticas y técnicas ágiles anteriormente descritas, para que los líderes de los equipos de desarrollo pudieran aplicarlas controlando cómodamente el estado del proyecto en todo momento.

También podría facilitar mucho la inclusión de distintos puntos de vista del cliente y comentarios sobre las tareas que se desarrollan, e incluso servir de plataforma de gestión de pruebas para la depuración y mejora del producto que se desea obtener, facilitando la comunicación del cliente con los desarrolladores.

De esta forma, también se puede reducir la necesidad de la presencialidad del cliente en todas las reuniones de seguimiento, pudiendo delegar sus funciones a través de este sistema automatizado de gestión.

Bibliografía

[Agi13] Agile Manifesto. [en línea] http://agilemanifesto.org [Consulta: 11 de Mayo de 2013]

[Alo05] ALONSO, Fernando; MARTINEZ, Loïc; SEGOVIA, Francisco Javier. *Introducción a la Ingeniería del Software*. 1ª Ed. Madrid: Delta Publicaciones Universitarias, 2005. 542 p. ISBN: 84-96477-00-2

[And03] ANDERSON. *Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results*. Prentice Hall, 2003. ISBN: 0-13-142460-2

[Bec02] BECK, Kent. *Test-Driven Development by Example*. 1ª Ed. Addison-Wesley Professional, 2002. 240 páginas. ISBN: 978-0321146533

[Ble10] BLÉ JURADO, Carlos y colaboradores. *Diseño ágil con TDD* [PDF]. Enero 2010. Disponible en web: http://www.dirigidoportests.com/wp-content/uploads/2009/12/disenoAgilConTDD.pdf [Consulta: 16 de Febrero de 2013]

[Boe88] BOEHM, B. W. *A Spiral Model of Software Development and Enhancement*. ACM Software Engineering Notes, vol. 11, nº 4, págs. 22-42, 1988.

[Fer13] FERNÁNDEZ-MEDINA PATÓN, Eduardo. *Ciclo de vida del Software* [PDF]. Disponible en web http://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI/Tema03.pdf [Consulta: 17 de Febrero de 2013]

[Has03] HASSAN MONTERO, Yusef; MARTÍN FERNANDEZ, Francisco J. *Más allá de la Usabilidad: Interfaces 'afectivas'* [en línea]. No Solo Usabilidad, nº 2, 2003. ISSN 1886-8592. Disponible en web http://www.nosolousabilidad.com/articulos/interfaces_afectivas.htm [Consulta: 3 de Julio de 2013]

[Hin13] HINCKLEY, Ken; JACOB, Robert J. K.; WARE, Colin. *Input/Output devices and interaction techniques* [PDF]. Disponible en web http://research.microsoft.com/en-us/um/people/kenh/papers/crc_iochapter.pdf

[Ino13] Prototipado. [en línea] http://www.inovdesigns.com/prototypes.php [Consulta: 12 de Mayo de 2013]

[ISO10] *ISO 9241-210:2010 - Ergonomics of human-system interaction – Part 210: Human-centred design for interactive systems*. International Organization for Standarization, 2011.

[Jos08] JOSKOWICZ, José. *Reglas y prácticas en eXtreme Programming* [PDF]. Febrero 2008. Disponible en web: http://www.uls.edu.sv/index.php?option=com_phocadownload&view=category&download=90%3Areglas-y-prcticas-en-programacin-extrema&id=2%3Ainformatica&ei=3dMfUYb1K8nRhAe6rIDoDA&usg=AFQjCNGcFPirwGae7gICBGsGzp14L_u6-g&bvm=bv.42553238,d.d2k&cad=rja [Consulta: 16 de Febrero de 2013]

[Ker99] KERNIGHAN, Brian Wilson; PIKE, Robert. *The Practice of Programming*. Reading, Massachusetts, Addison-Wesley, 1999. ISBN: 978-0201615869

[Nie13] Nielsen Norman Group. [en línea] http://www.nngroup.com/articles/paper-prototyping/ [Consulta: 12 de Mayo de 2013]

[Nor90] NORMAN, Donal A. *La psicología de los objetos cotidianos*. Donostia-San Sebastián, Editorial Nerea S.A., 1990. ISBN: 84-89569-18-5

[Pal07] PALACIO, Juan. *Flexibilidad con Scrum. Principios de diseño e implantación de campos de Scrum* [PDF]. Octubre 2007. Disponible en web: http://www.safecreative.org/work/0710210187520 [Consulta: 16 de Febrero de 2013]

[Pro13] Prototypes. [en línea] http://www.usability.gov/methods/design_site/prototyping.html [Consulta: 12 de Mayo de 2013]

[Sch11] SCHWABER, Ken; SUTHERLAND, Jeff. *The Scrum Guide - The definitive Guide to Scrum: The Rules of the Game* [PDF]. Disponible en web http://www.scrum.org/Portals/0/Documents/Scrum%20Guides/Scrum_Guide.pdf [Consulta: 7 de Abril de 2013]

[UC3M13] UNIVERSIDAD CARLOS III DE MADRID. *Cómo citar bibliografía* [PDF]. Disponible en web: http://www.uc3m.es/portal/page/portal/biblioteca/aprende_usar/como_citar_bibliografia/ [Consulta: 16 de Febrero de 2013]

[Wil04] WILLIAMS, Laurie. *Software Reviews and Pair programming* [PDF]. Disponible en web: http://agile.csc.ncsu.edu/SEMaterials/ReviewsandPairProgramming.pdf [Consulta: 10 de Mayo de 2013]

Índice alfabético